EMPIRICAL ASSESSMENT OF UML CLASS DIAGRAM LAYOUTS
BASED ON ARCHITECTURAL IMPORTANCE

A dissertation submitted
to Kent State University in partial
fulfillment of the requirements for the
degree of Doctor of Philosophy

by

Bonita Sharif

May 13, 2010

Dissertation written by

Bonita Sharif

B.Sc., Cyprus College, Nicosia, Cyprus, 1997

M.S., Kent State University, USA, 2003

Ph.D., Kent State University, USA, 2010

Approved by

____Dr. Jonathan I. Maletic _____, Chair, Doctoral Dissertation Committee

____Dr. L. Gwenn Volkert_____, Members, Doctoral Dissertation Committee

____Dr. Robert A. Walker_____

____Dr. Michael L. Collard_____

____Dr. Jocelyn R. Folk _____

Accepted by

____Dr. Robert A. Walker _____, Chair, Department of Computer Science

____Dr. Timothy Moerland_____, Dean, College of Arts and Sciences

TABLE OF CONTENTS

LIST OF FIGURES	XIII
LIST OF TABLES	XVIII
ACKNOWLEDGEMENTS	XXII
DEDICATION.....	XXIV
CHAPTER 1 INTRODUCTION.....	1
1.1 Motivation and Problem	3
1.2 Research Focus	6
1.3 Research Contributions.....	8
1.4 Organization of the Dissertation	9
1.5 Publication Notes	10
CHAPTER 2 RELATED WORK.....	11
2.1 Graph-based Aesthetics	11
2.1.1 Graph based Software Visualization Tools.....	16
2.1.2 Empirical Studies	18
2.2 UML Class Diagram Layouts	21
2.2.1 UML Style Guidelines	21
2.2.2 Algorithms and Guidelines	23
2.2.3 Colors.....	29
2.2.4 Metrics, Views, and Areas of Interest on Diagrams	31
2.2.5 Focus + Context	34

2.3	Empirical Studies on UML Class Diagram Layouts.....	36
2.3.1	Aesthetics.....	36
2.3.2	Stereotype Usage	40
2.4	Eye-tracking Studies	44
2.4.1	Usability Studies	45
2.4.1.1	Web Sites	46
2.4.1.2	Computer Based Tasks	49
2.4.1.3	Eye Trackers as Input.....	51
2.4.2	Software Engineering Studies.....	52
2.4.2.1	Source Code Comprehension.....	53
2.4.2.2	UML Class Diagrams	56
2.4.2.3	Program Visualizations	59
2.4.2.4	Program Debugging	61
2.5	UML Modeling Tools	64
2.6	Discussion on Related Work.....	66
CHAPTER 3	STEREOTYPED UML CLASS DIAGRAM LAYOUTS	73
3.1	Orthogonal Layout	74
3.2	Multi-cluster Layout	74
3.3	Three-cluster Layout	75
3.4	Summary	75
CHAPTER 4	THE PILOT STUDY	80

4.1	Experiment Overview	80
4.2	Class Diagram Layouts Studied.....	81
4.3	Study Design.....	81
4.3.1	Comprehension/Quantitative Design	83
4.3.2	Preference/Qualitative Design	84
4.3.3	Subject System.....	84
4.3.4	Subjects	85
4.3.5	Running the Study.....	86
4.4	Results and Analyses	87
4.4.1	Distribution of all Answers: Comprehension Questions	87
4.4.2	Preference for Stereotypes using Color and Text Annotations.....	89
4.4.3	Preference for Comprehension or Aesthetics: Comparing Three Layout Schemes	90
4.4.4	Distribution of Maintenance Questions	92
4.4.5	Preference for Aesthetics: Highlighted Regions and Curved Connectors	93
4.5	Summary	95

CHAPTER 5 THE EFFECTS OF LAYOUT ON UML NOTATION AND

DESIGN TASKS: REPLICATING AN EYE-TRACKING STUDY..... 97

5.1	Overview of the Original Eye-tracking Study by Yusuf et al.....	97
5.1.1	An Overview of the Study Design	98
5.1.2	Results and Conclusions from Yusuf et al.	99
5.2	Focus of the Study	101

5.3	Experimental Design.....	102
5.3.1	Experimental Goal and Hypotheses.....	102
5.3.2	Subject System.....	104
5.3.3	Tasks	104
5.3.4	Data Collection	108
5.3.5	Subjects.....	108
5.3.6	Study Instrumentation.....	110
5.4	Experimental Results and Analyses.....	111
5.4.1	Subject Classification.....	112
5.4.2	Question Classification	115
5.4.3	Effect of Layout on UML Tasks.....	119
5.4.4	Effect of Layout on Design Tasks	120
5.4.5	Effect of Ability and Layout on UML and Design Tasks.....	122
5.4.6	Confidence for Layouts in Subject Categories	124
5.4.7	Debriefing Questionnaire Results	125
5.5	Discussion	126
5.6	Similarities and Differences with the Eye-tracking Study by Yusuf et al.	128
5.7	Threats to Validity	129
5.8	Summary	129

CHAPTER 6 THE EFFECTS OF LAYOUT IN SOFTWARE

COMPREHENSION TASK CATEGORIES..... 131

6.1	Overview and Research Questions	131
-----	---------------------------------------	-----

6.2	Stereotyped Layouts.....	133
6.3	Experimental Design.....	133
6.3.1	Experimental Goal and Hypotheses.....	133
6.3.2	Subject Systems	136
6.3.3	Task Categories.....	136
6.3.4	Data Collection	140
6.3.5	Study Subjects.....	141
6.3.6	Running the Experiment	141
6.4	Experimental Results and Analyses.....	143
6.4.1	Effect of Layout on Accuracy.....	145
6.4.2	Effect of Layout on Speed	147
6.4.3	Secondary Factor Interactions.....	148
6.4.4	Confidence Level	150
6.4.5	Debriefing Questionnaire.....	151
6.4.6	Preference Ratings on Two Comprehension Tasks	151
6.5	Threats to Validity	153
6.6	Summary	154
 CHAPTER 7 A QUESTIONNAIRE-BASED STUDY ON DETECTING THE		
ROLE OF DESIGN PATTERNS		155
7.1	Design Patterns and Motivation.....	155
7.2	Class Diagram Layouts	157
7.3	Experimental Design.....	158

7.3.1	Subjects	161
7.3.2	Subject Systems And Design Pattern Selection.....	161
7.3.3	Comprehension and Preference Questions	162
7.4	Experimental Results	164
7.4.1	Accuracy and Time	164
7.4.2	Secondary Factor Interactions.....	166
7.4.3	Preference Ratings	170
7.4.4	Observations	170
7.5	Threats to Validity	171
7.6	Prior Work on Design Pattern Education.....	172
7.7	Summary	173

CHAPTER 8 AN EYE-TRACKING STUDY ON DETECTING THE ROLE OF DESIGN PATTERNS..... 174

8.1	Experimental Design.....	174
8.1.1	Eye-tracking Apparatus	175
8.1.2	Stimuli.....	176
8.1.3	Comprehension Task	178
8.1.4	Study Variables.....	178
8.1.5	Participants.....	184
8.1.6	Instrumentation	184
8.2	Experimental Results	185
8.2.1	Accuracy and Time	185

8.2.2	Visual Effort.....	188
8.2.3	Qualitative Analysis.....	195
8.3	Discussion.....	202
8.4	Threats to Validity	204
8.5	Summary	206
 CHAPTER 9 AN EYE-TRACKING STUDY ON THE EFFECT OF		
IDENTIFIER STYLES ON COMPREHENSION		207
9.1	Motivation.....	207
9.2	Overview.....	208
9.3	Research Questions.....	210
9.4	Experimental Design.....	211
9.4.1	Eye-tracking Apparatus	212
9.4.2	Material and Stimuli	212
9.4.3	Variables	214
9.4.4	Hypotheses	219
9.4.5	Participants.....	220
9.4.6	Instrumentation	221
9.5	Experimental Results	222
9.5.1	Correctness and Find Time	222
9.5.2	Visual Effort.....	226
9.5.3	Object Memory Task	230
9.5.4	Similarities and Differences.....	231

9.6	Discussion	232
9.7	Threats to Validity	233
9.8	Prior Work	235
9.8.1	Identifier Names.....	235
9.8.2	Research on Reading.....	236
9.9	Summary and Future Work.....	236
CHAPTER 10	GUIDELINES AND STATEMENTS BASED ON EMPIRICAL EVIDENCE	238
CHAPTER 11	CONCLUSIONS AND FUTURE WORK.....	244
 APPENDICES		
APPENDIX A RELATED WORK OVERVIEW		248
APPENDIX B STUDY MATERIAL FOR CHAPTER 4.....		250
B.1.	Comprehension Questions.....	250
B.2.	Preference Questions.....	252
B.3.	Diagrams in Orthogonal Layout.....	255
B.4.	Diagrams in Multi-cluster Layout	261
B.5.	Diagrams in Three-cluster Layout.....	267
APPENDIX C STUDY MATERIAL FOR CHAPTER 5		272
C.1.	UML Notation Questions	272
C.2.	High-Level Design Questions	274

C.3. Diagrams in Orthogonal Layout.....	277
C.4. Diagrams in Multi-cluster Layout	286
C.5. Diagrams in Three-cluster Layout.....	295
APPENDIX D STUDY MATERIAL FOR CHAPTER 6	304
D.1. Comprehension Questions for the Qt system.....	304
D.2. Comprehension Questions for the wxWidgets system	309
D.3. Preference Ratings Questionnaire	314
D.4. Qt Diagrams	315
D.5. wxWidgets Diagrams	331
APPENDIX E STUDY MATERIAL FOR CHAPTER 7.....	347
E.1. JHotDraw Diagrams in Multi-cluster Layout.....	347
E.2. JHotDraw Diagrams in Orthogonal Layout	351
E.3. JUnit Diagrams in Multi-cluster Layout.....	355
E.4. JUnit Diagrams in Orthogonal Layout	357
E.5. Qt Diagrams in Multi-cluster Layout	359
E.6. Qt Diagrams in Orthogonal Layout	361
APPENDIX F STUDY MATERIAL FOR CHAPTER 8.....	363
F.1. Composite Pattern Diagrams	363
F.2. Observer Pattern Diagrams.....	365
F.3. Strategy Pattern Diagrams	367
F.4. Singleton Pattern Diagrams	369

REFERENCES.....	371
------------------------	------------

LIST OF FIGURES

Figure 1.1. A sample UML class diagram showing classes and relationships	3
Figure 1.2. Process of obtaining a UML class diagram via reverse engineering.....	5
Figure 2.1. Visio's layout produced for the sample class diagram in Figure 1. using the <i>Compact Tree</i> style for nodes, <i>Right then down</i> Direction for edges and <i>Shallow</i> depth.....	65
Figure 3.1. A class diagram consists of elements in the design(class) model with layout techniques applied to them. The shaded area shows the research focus.....	73
Figure 3.2. Orthogonal layout for PlotterBase detailed view	77
Figure 3.3. Multi-cluster layout for PlotterBase detailed view.....	78
Figure 3.4. Three-cluster layout for PlotterBase detailed view	79
Figure 4.1. Setup for the pilot study	82
Figure 4.2. Distribution of answers at the <i>expert</i> level across all 3 layouts	88
Figure 4.3. Distribution of answers at the <i>intermediate</i> level across all 3 layouts	88
Figure 4.4. Color and text annotation preference across participant levels	90
Figure 4.5. Comparison of three class diagram layouts for the purpose of system comprehension	91
Figure 4.6. Comparison of three class diagram layouts for the purpose of aesthetic preference.....	91
Figure 4.7. Results for seven maintenance questions at intermediate and expert levels .	92
Figure 4.8. Multiple clusters highlighted based on architectural importance	94

Figure 4.9. Number of participants who preferred curved connectors for dependencies and associations in aesthetics.....	94
Figure 5.1. Class diagrams for the <i>PythonWrappers</i> module shown in three different layouts: orthogonal, three-cluster and multi-cluster	106
Figure 5.2. Descriptive statistics of subjects' background for the Yusuf et al replication study.....	109
Figure 5.3. UML and Design task scores across all twenty nine subjects. The maximum possible score is 42 for UML and 18 for design. The subjects are first sorted by UML score and then by design score.....	112
Figure 5.4. Time (in minutes) taken to complete UML and Design tasks across all twenty nine subjects. The order of subjects is the same as Figure 5.3 to have a side by side comparison.....	113
Figure 5.5. Box plot of the answer distribution between UML and design tasks.....	116
Figure 5.6. The class diagram in three cluster layout used for question 7: Which classes are derived from Observer? This is best viewed in color.	118
Figure 5.7. Box plot of UML scores across layouts for the Yusuf et al. replication study	120
Figure 5.8 . Descriptive statistics of design scores across layouts for the Yusuf et al. replication study.....	122
Figure 5.9. UML task scores across layouts for the Yusuf et al. replication study	123
Figure 5.10. Design task scores across layouts combined into low (DA+DI) and high (DK+DE) design ability for the Yusuf et al. replication study	124

Figure 5.11. The average confidence level for UML and Design tasks at each ability level for the Yusuf et al. replication study	125
Figure 5.12. Box plot of debriefing questionnaire results for the Yusuf et al. replication study	126
Figure 6.1. Descriptive Statistics for accuracy and time across groups for the task categories study	146
Figure 6.2. Interaction between layout and design experience with respect to time for the task categories study	150
Figure 7.1. An Example of the Strategy pattern template. The class Stat plays the role of Context, SortImpl plays the role of Strategy and SortBubble and SortShell play the roles of Concrete Strategies	156
Figure 7.2. The Strategy pattern in <i>Qt</i> shown in both layouts. The roles of context, strategy and concrete strategy are performed by <i>QXmlContentHandler</i> , <i>QXMLReader</i> , and <i>QXMLSimpleReader</i> respectively.	160
Figure 7.3. Accuracy and time box plots for patterns across layouts for the questionnaire-based study	166
Figure 7.4. Interaction between Layout and Experience for the four patterns for each dependent variable i.e., accuracy, time and relevant classes	169
Figure 8.1. Stimulus for detecting the role of Observer in the JUnit System (Q2)	177
Figure 8.2. Stimulus with areas of interest overlaid using red rectangles. The top left shows the task definition AOI. Each class's bounding box is an AOI. A design pattern cluster AOI is shown in dotted lines and shaded yellow. The classes at both	

ends of this cluster form the design pattern classes AOI. All classes outside the design pattern cluster are part of the non-design pattern classes AOI.	180
Figure 8.3. Accuracy and time box plots for patterns across layouts for the eye-tracking study	188
Figure 8.4. Box plots for FC_T and AFD_T	190
Figure 8.5. Box plots for $FR_{DPCluster}$ and $AFD_{DPCluster}$	192
Figure 8.6. The average number of fixations and the average fixation duration for each layout across the four design patterns	194
Figure 8.7. A heat map based on fixation length of all eight experts for the Strategy pattern	196
Figure 8.8. A heat map based on fixation length of all seven novices for the Strategy pattern	197
Figure 8.9. A heat map based on fixation length of all fifteen subjects for the Observer pattern	198
Figure 8.10. A gaze plot of an expert for the Observer pattern	199
Figure 8.11. A gaze plot of a novice for the Observer pattern.....	200
Figure 8.12. A gaze plot of an expert for the Strategy pattern.....	201
Figure 8.13. Portion of a stimulus comparing an expert and novice gaze plot for Singleton pattern.	202
Figure 9.1. Phrase Stimulus (a) with task description in bold. Question Stimulus (b) with clouds to detect the correct identifier formed using the phrase	214

Figure 9.2. Descriptive statistics for Find Time in each category of identifier Style (CC=CamelCase and US=Under score)	223
Figure 9.3. Interaction between subjects' experience and identifier style.	224
Figure 9.4. Part of two gaze plots for the correct underscore and camel cased versions of the 3-word code identifier	228
Figure 9.5. Descriptive statistics for the Average Fixation Duration on correct identifiers (AFD _{correct})	229
Figure 9.6. Part of a gaze plot for a novice showing three large fixation durations on each of the three parts of the distracter <i>mouseTheaterTicket</i>	230

LIST OF TABLES

Table 2.1. Overview of graph aesthetics.....	13
Table 2.2. UML style guidelines with respect to layout	22
Table 4.1. Summary of participants in pilot study.....	85
Table 5.1. Experiment overview for Yusuf et al.'s replication.....	103
Table 5.2. Three design tasks asked for the PythonWrappers module. The layouts used for these questions are shown in Figure 5.1.....	107
Table 5.3. Debriefing Questionnaire.....	108
Table 5.4. Categorization of subjects for UML and Design tasks. UML and design tasks are shown by an U and D respectively.....	114
Table 5.5. UML and Design Question Classification	117
Table 5.6. Results of the pair wise Wilcoxon test for UML tasks. Significance is shown in bold. $\alpha = 0.05$. For the 1-tailed values, the direction is given by the order of treatment pairs i.e., first < second.	119
Table 5.7. Results of the pair wise Wilcoxon test for design tasks. Significance is shown in bold. $\alpha = 0.05$. For the 1-tailed values, the direction is given by the order of treatment pairs i.e., first < second.	121
Table 6.1. Experiment overview for task categories study.....	134
Table 6.2. A 2*2 factorial design with two systems and two layouts for the task categories study.....	135

Table 6.3. Modules per task category used in <i>Qt</i> and <i>wxWidgets</i> (wx). Each module is represented by a class diagram in two layouts: orthogonal (ortho) and multi-cluster (multi). The difficulty level is shown near each task category.	139
Table 6.4. Layout and system characteristics. The number of edge crossings and edge bends for each layout. The number of classes and relationships is the same across both layouts.	140
Table 6.5. Descriptive statistics for <i>accuracy</i> and <i>speed</i> in each task category for each system.	144
Table 6.6. Task categories study: 1-tailed <i>p-values</i> for the paired Wilcoxon test (alpha=0.05). Cohen's d denotes the effect size: 0.2(small), 0.5 (medium), >=0.8 (large). Alternative hypotheses for the 1-tailed tests indicate the following directionality: $\mu(\text{Accuracy}_{\text{ortho}}) < \mu(\text{Accuracy}_{\text{multi}})$ and $\mu(\text{Speed}_{\text{ortho}}) > \mu(\text{Speed}_{\text{multi}})$. * indicates significance.	145
Table 6.7. <i>Mann-Whitney</i> results (1-tailed) of preference ratings for 2 questions in each system for the task categories study.....	152
Table 7.1. Experiment overview for detecting design pattern roles study.....	159
Table 7.2. Overview of subject systems, classes and design patterns used	162
Table 7.3. Role detection questions in four patterns across three systems for the questionnaire-based study	162
Table 7.4. Questionnaire based study: 1-tailed Wilcoxon <i>p-values</i> (alpha=0.05) for accuracy, time, and relevance for each design pattern's role detection. Directionality	

implies that the multi-cluster layout performs better. Cohen's d denotes the effect size: 0.2(small), 0.5 (medium), ≥ 0.8 (large). * indicates significance.	164
Table 7.5. Results of the <i>ANOVA</i> test with the Experience factor. p-values for Accuracy (Time) are mentioned for each pattern. * indicates significance.	167
Table 7.6. Questionnaire-based study: <i>Mann-Whitney p-values</i> (1-tailed: multi-cluster > orthogonal, $\alpha=0.05$) for preference ratings. * indicates significance	170
Table 8.1. Role detection questions in four patterns across three systems for the eye-tracking study.....	178
Table 8.2. Eye-tracking study: 1-tailed Wilcoxon <i>p-values</i> ($\alpha=0.05$) for accuracy, time, and relevance for each design pattern's role detection. Directionality implies that the multi-cluster layout performs better. Cohen's d denotes the effect size: 0.2(small), 0.5 (medium), ≥ 0.8 (large). * indicates significance.....	187
Table 8.3. 1-tailed Wilcoxon <i>p-values</i> ($\alpha=0.05$) for the visual effort measures. Directionality implies the multi-cluster layout performs better.....	189
Table 9.1. Experiment overview for studying identifier styles.....	211
Table 9.2. Phrases used in the study.	213
Table 9.3. Null hypotheses for the identifier style study	219
Table 9.4. Model parameters for interaction between Experience and Style	224
Table 9.5. Complex Model parameters for Find Time	225
Table 9.6. Two-tailed Wilcoxon <i>p-values</i> ($\alpha=0.05$) for each dependent variable. CC=camel case, US=underscore.....	227
Table 10.1. An overview of the experiments conducted	238

Table 11.1. UML class diagram comprehension studies closely related to the research 248

ACKNOWLEDGEMENTS

I am extremely grateful to my husband, Sharif, whose support made this thesis possible to accomplish. He was always there to offer support and encourage me to finish this work. He kept me focused and grounded in many aspects. Thank you Sofia, for being my inspiration to complete this work and for being so understanding. I would especially like to thank my mother who encouraged me 16 years ago, to start my undergraduate studies in Computer Science. I would not be at this point, if it wasn't for her belief for me.

I am very thankful to my advisor, Dr. Jonathan Maletic, for believing in me. He was always available to offer guidance and point me in the right direction. I could not ask for a better advisor. I am thankful to Dale Haverstock, and Dr. Michael Collard for supporting the conducting of these studies in the classroom.

I would like to individually thank every single person who participated in the studies presented here. This work would not have been possible without your time. The eye-tracking studies done in this thesis could not have been done without the assistance of Dr. David Robbins, who was very helpful in letting me use the equipment.

I also want to thank my Dissertation Committee for being part of this process and offering valuable input. My sincere thanks to everyone in the Department of Computer

Science at Kent State University whose names I did not mention, but who contributed in any form towards the completion of the dissertation.

Bonita Sharif

May 2010, Kent, Ohio

DEDICATION

To my parents, Irene and Augusto Simoes

CHAPTER 1

Introduction

The Unified Modeling Language (UML) [Booch, Rumbaugh, Jacobson 2005] is widely used in various domains as the de facto standard graphical notation to represent user requirements and design of object oriented software systems. Booch defines UML as “*a graphical language for visualizing, specifying, constructing and documenting the artifacts of a software-intensive system*” [Booch 1999]. The UML provides a wide range of diagrams each contributing to a different view of a complex system. The dissertation focuses on the design view or structural/static view of software which is captured by the class diagram. UML class diagrams contain the main components of an object-oriented system. They present a set of classes, interfaces, collaborations and their relationships [Booch, Rumbaugh, Jacobson 2005]. Experimental evaluations conducted on systems using UML diagrams for analysis and design, have shown significant improvements in the correctness of changes and design quality versus those that do not use any type of modeling method [Arisholm, Briand, Hove, Labiche 2006; Tilley, Huang 2003].

A class is an abstraction of a set of entities from the domain of discourse sharing common structure and behavior. A class diagram can be viewed as a graph where each node represents a class and each link represents a relationship between classes. There are two main types of relationships: *generalizations* also known as inheritance hierarchies and *associations* that may be further categorized as an *aggregation* or *composition*. Other types of relationships include dependencies and realizations. The classes and

relationships have certain properties attached to them. Classes have attributes and operations. Relationships have roles and cardinalities associated with them.

Stereotypes are an extension mechanism provided by the UML that allow users to define semantics on the notation thereby extending the language [Gogolla, Henderson-Sellers 2002]. Wirfs-Brock [Wirfs-Brock 1994] shows the usefulness of using stereotypes for object classification with respect to the roles and responsibilities in the system. Stereotypes have been used to support the classification of classes and objects by assigning them some features and properties [Atkinson, Kuhne, Henderson-Sellers 2002]. In the analysis model of software, three different class stereotypes are used: *control*, *boundary* and *entity* [Booch, Jacobson, Rumbaugh 1999]. A class fits into one of these three basic stereotypes. A description of each stereotype is now given.

A *boundary* class models interaction between a system and the external world namely consisting of users and other external systems. Examples of boundary classes include those responsible for abstraction of forms and communication interfaces. An *entity* class models persistent information in a system. Entity classes capture some semantic structure in the business rules/logic. A *control* class models the coordination and sequencing of other objects in a system. Examples of control classes include those that monitor main events and control flows by designating work to boundary or entity classes. Additional details about these stereotypes can be found in [Booch 1999].

In a class diagram, stereotypes are surrounded by guillemets (« »). An example of a class diagram for a drawing application is shown in Figure 1.1.

Figure 1.1. A sample UML class diagram¹ showing classes and relationships

1.1 Motivation and Problem

In object-oriented design and development, it is essential to know the main role and functionality of a class. It is important to detect system boundaries, entities and control tasks performed by the system and coordination between boundary and entity objects. Class stereotypes are important in the process of understanding the role and importance of each class in the system as well as the whole system. These stereotypes

¹ This diagram was constructed using concepts from the Unified Modeling Language User Guide by Booch et al. and the tutorial at <http://www.objectmentor.com/resources/articles/umlClassDiagrams.pdf>

determine the architectural importance of classes in a system. The research presented uses these stereotypes in the layout of UML class diagrams.

The usefulness of class diagrams can be approached from two perspectives: during initial system design creation and during software maintenance of an existing system. The work presented here emphasizes the latter. During the initial stages of building a new system (forward engineering), the class model is constructed using several class diagrams that help capture system features and user requirements. During software evolution (due to changes in user requirements or bug fixes), class diagrams help in understanding existing complex systems. Before adding or modifying features in a system, it is vital to have an accurate mental model [Littman, Pinto, Letovsky, Soloway 1986] of the system. This becomes even more important when people who maintain the system are usually not the ones who originally developed it. In this dissertation, the focus is on supporting software maintenance activities with stereotyped class diagram layouts.

During software maintenance, a reverse engineering tool like the ones presented in [Guéhéneuc 2004; Guéhéneuc, Albin-Amiot 2004; Keschenau 2004; Kollman, Selonen, Stroulia, Zündorf 2002; Matzko et al. 2002; Sutton, Maletic 2007; Tonella, Potrich 2001] are typically used to generate UML class models in order to get an overview of the interactions between classes in the system. UML class diagrams are then drawn based on these reverse engineered UML class models conforming to a particular layout algorithm. Figure 1.2 illustrates this process.

The generation of UML class diagrams from reverse engineered UML class models is not necessarily automatic. Commercial tools such as Visual Paradigm for UML [VP 2007] automatically generate very rudimentary UML class diagrams after reverse engineering the UML class model. However, they are very difficult to comprehend since they usually contain all (more than needed) classes and relationships from the class model laid out in a haphazard way. Large systems typically consist of thousands of classes which result in a hodgepodge of classes and relationships.

Figure 1.2. Process of obtaining a UML class diagram via reverse engineering

Class diagrams of large systems can be quite complex causing problems in readability and comprehensibility of large systems [Koning, Dormann, van Vliet 2002]. Relationships between elements in a UML diagram can also be unclear if poorly presented. In 1956, Miller [Miller 1956] showed that there is a limit to how much

information we can process at a time. This is visually limited to 7 ± 2 elements. Current UML diagrams do not present UML models in manner that is easy to understand the underlying system [Lange, Chaudron 2007; Tilley, Huang 2003]. There is still a need to represent UML models in a way that makes system comprehension easier.

According to [Booch, Rumbaugh, Jacobson 2005], the purpose of a UML diagram is not to draw pretty pictures but to visualize, specify, construct and document. An aesthetically pleasing class diagram might look pretty but it increases the cognitive workload of the person reading it. A variety of UML class diagram tools [Eichelberger 2001; Eiglsperger, Kaufmann, Siebenhaller 2003; Gutwenger et al. 2003b] have been developed but relatively little effort is put into evaluating the effectiveness and utility of the tools. This work bridges the gap in empirical evaluation of UML class diagram layout techniques.

1.2 Research Focus

The focus of the research is the empirical validation of different UML class diagram layouts based on architectural importance in the context of system comprehension. The ultimate purpose of the research is to determine effective ways to adjust the layout of existing UML class diagrams to support program comprehension. This is referred to as layout adjustment. In order to realize this, a family of experiments are designed and conducted with respect to different types of software maintenance tasks. Replication with different methods of data collection are used. Two of the experiments are replicated using an alternate method of data collection to support and verify the findings. When an online questionnaire-based study is replicated using eye-tracking

equipment, a more fine-grained analysis is presented. When an eye-tracking study is replicated using online questionnaires, additional verification and support for the eye-tracking study is obtained. The research shows that traditional questionnaire-based methods complement eye-tracking studies.

The high level hypothesis is that laying out a class diagram based on class roles and architectural importance of class stereotypes will result in higher accuracy and faster system comprehension thereby reducing the effort needed to perform the task. The main research questions the dissertation addresses are:

- RQ1: Is there an improvement in the comprehension of software maintenance tasks for stereotyped class diagram layouts vs. layouts based on pure aesthetics?
- RQ2: Which software comprehension tasks benefit most from stereotyped class diagram layouts?
- RQ3: Do stereotyped layouts help design experts and novices in the same way?
- RQ4: Does the layout of a class diagram affect the visual effort needed during a software maintenance task?
- RQ5: Does the layout of a class diagram affect the eye gaze behavior of experts and novices? What are the similarities and differences?

Each of these research questions are discussed in the context of the studies conducted herein. A set of four studies are conducted to address the first three research questions. An eye-tracking study is conducted to address the last two questions. In addition, an empirical study on identifier styles (camel case and underscore) is conducted to determine the effect identifier styles have on readability. Since UML class diagrams

contain identifiers as data members, this study is seen as complementary empirical evidence to the layout used.

1.3 Research Contributions

The research makes the following contributions to the empirical assessment (qualitative and quantitative) of layout schemes for UML class diagrams with respect to program comprehension.

1. An empirical methodology assessing the effectiveness of different class diagram layout schemes with respect to different software task categories.

The task categories involve:

- a) UML notation and high level software design tasks
 - b) Six software maintenance tasks (Reading, Overview, Impact Analysis, Bug Fix, Feature Addition and Refactoring)
 - c) Design pattern role detection tasks. This has direct impact on the education aspect of design patterns with layout playing a major role.
2. The replication of two studies using eye-tracking and/or online questionnaires to determine added benefits using either approach. A comparison between replications of eye-tracking and online questionnaire studies provides additional evidence in support of class diagram layout schemes.
 3. A set of quantifiable eye tracking measures to determine UML class diagram layout effectiveness in solving a task.
 4. An empirical study to determine the effect of identifier style on the readability of UML class diagrams.

5. A set of guidelines and statements based on the empirical evidence presented.

This information will directly aid in creation of an algorithm to adjust the layout of existing UML class diagrams.

The main contribution is the user-centric empirical investigation of UML class diagram layout techniques based on architectural importance (points 1,2, and 3). Both traditional and eye-tracking methods were used in the investigation. Instead of developing a tool that uses stereotype-based layout techniques, the approach taken is to first empirically validate the techniques that are most useful and base the tool on the empirical evidence. A secondary contribution (point 5) is a set of empirically validated guidelines based on the semantics of classes and their roles in a software system. This is to be realized in a future class diagram layout adjustment tool. The fourth point assesses the readability aspect of UML class diagrams that is viewed as complementary to layout schemes.

1.4 Organization of the Dissertation

The remainder of the dissertation is organized as follows. Chapter 2 discusses related work in the area of UML class diagram layouts, relevant empirical studies, and eye-tracking studies. Chapter 3 introduces the three layout schemes used in the family of experiments conducted in this thesis. Chapter 4 presents the initial pilot study that seeks to evaluate the feasibility of the approach. Chapter 5 replicates an previous eye-tracking study using online questionnaires and compares and contrasts the findings. Chapter 6 presents a controlled experiment examining the effect of layout in six different software maintenance task categories. Chapter 7 studies the effect layout has in detecting the role

of design patterns in class diagrams. Chapter 8 replicates the previous experiment using an eye-tracker using a slightly different setup. Chapter 9 presents an eye-tracking study on the effect of different identifier styles on the readability aspect of UML class diagrams. Related work on identifier styles is also presented here. Based on the empirical studies conducted, Chapter 10 summarizes main guidelines to be realized in a class diagram layout adjustment tool. Conclusions and future work are presented in Chapter 11.

1.5 Publication Notes

Portions of this dissertation are extended versions of previously published papers. Chapter 4 is reflected in the work presented in [Andriyevska, Dragan, Simoes, Maletic 2005]. Chapter 5, Chapter 6, and Chapter 7 have been published at the 17th International Conference on Program Comprehension (ICPC 2009) [Sharif, Maletic 2009b], 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2009) [Sharif, Maletic 2009a], and at the 23rd IEEE-CS International Conference on Software Engineering Education and Training (CSEE&T 2010) [Sharif, Maletic 2010a] respectively. Chapter 9 is published at the 18th International Conference of Program Comprehension (ICPC 2010) [Sharif, Maletic 2010b].

CHAPTER 2

Related Work

The chapter presents a detailed literature review on graph aesthetics and UML class diagram layouts. The methodology used is described, including research questions and validation of results. First, important graph based aesthetics are discussed along with their applications in software visualization. A number of methods that propose new criteria including layouts for UML class diagrams follows. Next, empirical studies conducted on UML layouts and on domain specific stereotypes in class diagrams are discussed. Eye tracking studies done in different settings including software artifacts such as source code and UML class diagrams is described next. Finally, layouts available in some popular commercial and open source tools are briefly discussed. The chapter concludes with a summary and discussion of the approaches.

2.1 Graph-based Aesthetics

We now formally define what we mean by a graph. A graph $G = (V, E)$ consists of a finite set of vertices (or nodes) denoted by V connected by a finite set of edges denoted by E . An edge $E = \{(u, v) \mid u, v \in V\}$ consists of a link between two vertices from V . A graph is connected if there is a path between any two vertices of the graph. A multi-graph is a graph² where multiple edges are allowed between any two vertices. A

² The terms graph and connected graph may be used interchangeably.

UML class diagram $U = (C, R)$ is a connected multi-graph where C represents classes (vertices) and R represents relationships (edges) between classes. A UML class diagram consists of both directed and undirected relationships.

This section discusses general graph based aesthetics that influence UML class diagrams. Several graph drawing algorithms have been proposed in the literature. Battista et al. [Battista, Eades, Tamassia, Ioannis 1999; Battista, Eades, Tamassia, Tollis 1994] is an excellent source for these algorithms. The main types of graph drawings are polyline drawings, straight line drawings, orthogonal drawings, grid drawings and planar drawing. In a polyline drawing, the edges are drawn using a polygon chain. A straight line drawing uses straight lines as connecting edges. An orthogonal drawing is drawn as a polygon chain of alternating horizontal and vertical line segments. In a grid drawing, nodes and edges conform to an integer grid. A planar drawing does not have edge crossings. Orthogonal drawings are used mainly in software engineering such as in UML class diagrams. While drawing graphs, aesthetics are used to make graph reading easier. The main aesthetic criteria for general graphs are given in Table 2.1.

These aesthetics frequently conflict with each other resulting in tradeoffs. The aesthetics mentioned in Table 2.1 are computationally intensive which results in a set of heuristics and approximations of them. Many of the heuristics given are NP-hard. For example, minimizing the number of bends in orthogonal planar drawings is NP-hard. Most of the orthogonal drawing construction algorithms run in $O(n + m)$ where n is the number of nodes and m is the number of edges.

Table 2.1. Overview of graph aesthetics

Minimize	<ul style="list-style-type: none"> ▪ total number of edge crossings ▪ area of the drawing ▪ total edge length ▪ maximum length of an edge ▪ large differences between edge lengths ▪ bends ▪ maximum number of bends ▪ large differences between number of bends on edges ▪ aspect ratio of the drawing
Maximize	<ul style="list-style-type: none"> ▪ smallest angle between two edges incident on the same vertex ▪ symmetry

There are also a set of constraints associated with graphs which involve positioning of nodes. Some nodes might be constrained towards the boundary and others might be required to be placed towards the center. The three most commonly used approaches to drawing graphs are the *topology-shape-metrics* approach, *hierarchical* approach and the *force-directed* approach. We discuss these briefly.

The *topology-shape-metrics approach* is taken to construct orthogonal grid drawings where an edge is mapped onto horizontal and vertical segments on a grid. It is based on three equivalence properties, topology, shape and metrics. The topology of two orthogonal drawings is the same if they can be obtained from each other. The shape of two orthogonal drawings is the same if they have the same topology and the lengths of edges can be modified without changing the angles and finally if two orthogonal drawings are congruent up to a translation or rotation they have the same metrics. The three steps involved in drawing graphs using a topology-shape-metrics approach are *planarization*, *orthogonalization*, and *compaction*. The planarization step eliminates

edge crossings. The orthogonalization step gives shape to the graph by imposing angles for each edge. The minimization of edge crossings aesthetic is satisfied here. The compaction step tries to minimize the area of the graph from the orthogonalization step. The compaction step also satisfies other aesthetics such as minimizing sum of the lengths of edges. Each of these steps in the topology-shape-metrics approach can also have a set of constraints attached to them.

The *hierarchical approach* is used for directed graphs. These graphs are drawn using three steps namely the layer assignment, the crossing reduction and the x-coordinate assignment. After the first step of assigning layer numbers to vertices, the crossing reduction step assigns order to vertices and minimizes crossings. The last step gives the x coordinates for each of the vertices that were laid out vertically using the layer assignment step.

The *force-directed* approach is a method to create straight line drawings of undirected graphs. The idea behind these drawings is to simulate forces on a graph with the goal of finding a locally minimum energy configuration. The two aspects of this layout approach are the force model and the energy configuration. Typically a spring is used as the force to be applied to nodes. Repulsive and attractive forces are applied between graph elements so that their layout reaches a state that minimizes these forces. These algorithms produce symmetrical drawings and distribute nodes evenly.

Herman et al. provides an excellent survey of graphs in the context of information visualization [Herman, Melancon, Marshall 2000]. Diaz et al. present a survey of graph layout problems [Diaz, Petit, Serna 2002] some of which are applicable to class diagrams.

An experimental comparison of four graph drawing algorithms was done in [Battista 1997]. Sugiyama presents a set of steps that address the automatic layout of graphs from a cognitive viewpoint [Sugiyama 1987]. A constrained based approach to graph drawing is given in [Ryall, Marks, Shieber 1997]. It is an interactive approach that lets a user change the graph while maintaining the constraints. Some of the constraints they deal with are node-node overlap, node-edge overlap, alignment, clustering and symmetry. Methods for grouping nodes in graphs within clusters using a force-directed layout is given in [Six, Tollis 2001].

A scalable force-transfer algorithm to overcome node overlaps in graphs is presented in [Huang, Sajeev, Lai 2006]. They claim that current algorithms do not consider node size and hence do not work well. Uniform scaling results in a large expanded layout. Constrained optimization is very slow and force-based algorithms use an iterative approach causing scalability problems. Given a graph with overlapping nodes, they adjust the layout and remove node overlaps given a minimum threshold between nodes. The authors' state that this approach can be used in UML diagrams and refers to them as labeled graphs. However, they do not state how this can be done in a concrete way. Lai et al. present a case as to why layout adjustment is needed in the case of practical graphs such UML class diagrams [Lai, Eades 2002] that do not conform to typical node-link graphs. Their goal is to first apply an abstract graph layout algorithm to the UML class diagram and second, to remove edge-node intersections and overlapping edges using a force scan algorithm. Harel et al. consider node size in their spring

embedded algorithm to prevent node overlapping vertices [Harel, Koren 2002]. These are related to abstract graphs and not UML class diagrams.

2.1.1 Graph based Software Visualization Tools

An overview of the types of software visualization tools that use graphs as their underlying structure is presented next. Reiss presents a software visualization tool *Bloom* [Reiss 2001] that helps in software comprehension. It supports queries to filter the output. The graphs are a combination of static and dynamic aspects of program data. One such example is a dynamic call graph that uses trace data and structural information from the program. Fronk et al. [Fronk 2006] visualize source code structures in 3D. Instead of using UML diagrams, they use tool *VisMOOS* that selectively displays parts of code using different layout criteria for comprehension. Sawant et al. use an iconic based method to visualize software architectures [Sawant, Bali 2007]. The architecture is constructed using static analysis.

Rigi [Wong, Tilley, Muller, Storey 1995] is a graph visualization tool that helps in visualizing large data structures such as software programs and documentation. It extracts relevant artifacts from the information source and presents them at a higher level of abstraction for easier comprehension. It is mainly graph based. A hierarchical method of clustering is used to group elements that have some common properties for e.g., sub-systems in software. There are several built-in metrics for graph filtering such as coupling and cohesion within sub-systems. A scripting mechanism is also provided for extendibility. SHriMP [Storey, Wong, Fracchia, Müller 1997] is a software visualization tool that supports multiple perspectives/views using a nested graph display. It supports

zooming, fish eye and filtering mechanisms. Several graph layout techniques [Storey, Müller 1995] are used in the SHriMp environment. Fisheye views are used for nested graphs. Nodes are adjusted on the screen when the focus of the graph changes. All nodes are scaled to fit inside the display area.

Storey et al. also discuss a hierarchy of cognitive design elements [Storey, Fracchia, Müller 1997] that need to be considered in software exploration tools. They distinguish between a mental and cognitive model where a cognitive model describes cognitive processes and input used to form the mental model. The hierarchy consists of strategies programmers use during comprehension and the cognitive overhead of navigating through software. The strategies programmers use fall under three categories, bottom-up, top-down and integrate bottom-up and top-down. The cognitive overhead categories involve facilitating navigation and providing orientation cues such as the current focus.

Lanza et al. present a unique visualization technique that presents the internal structure of a class called the class blueprint [Lanza, Ducasse 2001]. Graphically, a class is split into five sections; initialization, interface, implementation, accessor, and attributes. Relationships between elements of these five sections are shown based on the code structure. The class blueprint helps a software engineer to get a quick high level view of the workings of a class without going into the code level details.

Irwin et al. present a visualization technique called class clusters [Irwin, Churcher 2003]. Class clusters are based on the idea that semantically closely related classes

should be physically close. This property is generally not considered in diagramming tools.

Price et al. present a broad taxonomy of software visualization [Price, Baecker, Small 1993] based on several tools available. Empirical studies based on graphs are described next.

2.1.2 Empirical Studies

Purchase [Purchase 1997] presented a study to evaluate the importance of five graph aesthetics. The aesthetics involved were to minimize bends, crossings, maximize the minimum angle between two adjacent edges leaving a node, and maximize orthogonal edges and symmetry. The goal of this study was to prioritize the aesthetics based on their importance in human understanding. The experiments were conducted online and measured the ease of understanding based on time and accuracy of answers. Subjects were asked questions about several different drawings of the same graph conforming to each of the five aesthetic criteria. In particular, ten handcrafted experimental graph drawings were each displayed three times, once for each question. The graphs had 16 nodes and 28 edges. One of the questions in the study asked the participants to determine how many edges and nodes need to be removed in order to disconnect two given nodes. Another question was asked about the shortest path length between two nodes. The results show that reducing edge crossings was the most important aesthetic. Minimizing the number of bends and maximizing symmetry had a lesser effect. The angles and orthogonal edges did not play a statistically significant role

in the results obtained. A similar paper-based experiment [Purchase, Cohen, James 1997] was conducted with comparable results.

In another study, Purchase et al. [Purchase 1998b] study eight different graph layout algorithms with respect to one graph. The experiment consisted of subjects answering certain questions about eight graph drawings of the same graph. Specific layouts were chosen from the literature and implemented in the *GraphEd* tool. This experiment was solely based on graph reading and was not related to a task in an application. Time and accuracy were recorded. The questions were the same as their earlier experiment [Purchase 1997]. The results indicate similar accuracy between subjects on all algorithms for the graph with one exception. The *Seisenberger* algorithm was significantly more difficult to comprehend than the two force-directed and incremental algorithms used. The response time for the subjects to answer the questions was within the same range. In general the results were inconclusive making it difficult to state the superiority of one algorithm in particular. The results of both the experiments above are summarized in [Purchase 1998a; 2000].

Ware et al. [Ware, Purchase, Colpoys, McGill 2002] conducted an experiment that measured the cognitive cost of graph aesthetics. They evaluate the following graph layout criteria; edge crossings, sum of the lengths of the edges, path bendiness (edge continuity) and number of branches emanating from nodes on a path. The experiment involved finding shortest paths in graphs with path length between three and five. The results indicate that the length of paths, edge continuity, edge crossings, and number of branches emanating from a node were the most important factors to affect the cognitive

cost. The continuity aesthetic was the most significant result. These results apply only to graphs where the task involves finding shortest paths.

A set seven metrics to gauge the aesthetic quality of abstract graphs are presented as a result of the many experiments conducted by Purchase [Purchase 2002]. The measurement is in the form of a real number between 0 and 1 where 1 determines the positive effect of the aesthetic. In particular, metrics based on objective measures are defined for minimizing edge crossings and edge bends and maximizing symmetry, minimum angle between edges leaving a node, orthogonal edges, orthogonal nodes, and, consistent flow direction. The metrics are applied on a set of graphs of varied sizes. They found the value of the edge crossing aesthetic to be high since all the algorithms tried to minimize the number of edge crossings. The presented metrics will help formally quantify output produced by different algorithms.

Purchase et al. also conducted an experiment on a dynamic graph algorithm and determined the effect on the user's mental model [Purchase, Hoggan, Gorg 2006]. A dynamic graph is one that changes over time. One method is to use the static layout algorithms discussed above and apply them when a change occurs. The two criteria that need to be considered are readability of the layouts used and mental map preservation. The readability depends on the aesthetic criteria and mental map is preserved by making sure nodes appear in the same positions so that they may be easily identified. The layout and mental map criteria can contradict each other since the layout might want to change the position of an existing node whereas the mental map criterion states that it should maintain its position. The goal of this paper is to determine from a user's standpoint if it

is preferable to preserve the mental model, apply a static layout every time a graph changes or make a compromise between the two. The results indicate preserving of mental map for evolving graphs if nodes in the graph need to be referenced by name. It was not as important if tasks focused on edges.

2.2 UML Class Diagram Layouts

The following sections discuss layout techniques specific to class diagrams. UML style guidelines are described first, followed by various techniques that affect the layout of class diagrams.

2.2.1 UML Style Guidelines

Most of the guidelines in UML style [Ambler 2002] are related to the UML notation or to the design or analysis model being represented. For example, certain criteria might apply only to the design model whereas others might apply only to the analysis model. The analysis model is a higher level abstraction of the design model. These criteria are mainly related to the UML notation rather than the layout. Layout guidelines make up a small chunk in these style guides. Table 2.2 summarizes all guidelines related to layout described by Ambler.

The general guidelines are with respect to UML diagrams as a whole. Most of the general guidelines are borrowed from general graph aesthetics [Battista, Eades, Tamassia, Ioannis 1999] such as minimizing edge crossings, avoiding curved edges, using symmetry. The second general guideline says if two lines cross, a jump over the cross is needed to accurately show the connections. An example of a jump is shown between

TextPlotter and *TextDataRep* in Figure 3.2. The use of white space in diagrams is also recommended. Since we are used to reading from left to right and top to bottom in the English language, it is stated that the same method be used when organizing diagrams. The guidelines also state the use of color and stereotypes be limited and use of extra decorations such as font sizes, line sizes be used cautiously.

Table 2.2. UML style guidelines with respect to layout

General Guidelines

1. Avoid crossing lines.
2. Depict crossing lines with a jump.
3. Avoid diagonal or curved lines.
4. Use consistently sized symbols.
5. Arrange symbols symmetrically.
6. Use white space in diagrams.
7. Organize diagrams from left to right and top to bottom.
8. Organize large diagrams in several smaller ones.
9. Apply color and stereotypes albeit sparingly.
10. Apply decorations sparingly.

Relationship Guidelines

1. If an association is modeled by a class, center the class and connecting line between the two associated classes.
 2. Place subclasses below super-classes.
 3. In associations place the whole part to the left.
 4. Model inheritance vertically.
 5. Model all other relationships (associations, dependencies, realizations) horizontally.
 6. Use a “tree configuration” to draw two or more same-type relationships to a class.
-

Ambler also gives guidelines with respect to relationships. The table mentions only the ones related to layout specific to class diagrams. Most of these are common sense guidelines. For associations, it states that if a class is used to model the association, the placement should be in the middle of the two associated classes. For generalizations,

it states that derived classes should be placed below their parent. Associations should be laid out horizontally whereas generalizations should be laid out vertically. The third relationship guideline states that in an aggregation or composition, the whole part of the relationship should be to the left. This can be seen in Figure 1.1 between classes *Circle* and *Point* and between classes *Window* and *Shape*. The sixth relationship guideline states that a tree configuration should be used to draw relationships. This implies drawing one line to the class that joins all same-type relationships thereby reducing clutter. An example of such a tree configuration can be seen in the generalization relationship between the classes *Shape*, *Circle*, *Rectangle*, and *Polygon* in Figure 1.1. However, a warning states that sometimes using a tree arrangement is not always beneficial besides making the diagram look better.

2.2.2 Algorithms and Guidelines

This section presents several layout algorithms and guidelines for class diagrams presented in the literature. Many of these algorithms rely on graph aesthetics discussed in Section 2.1.

Eichelberger et al. present a layout algorithm that generates UML class diagrams from Java source code [Eichelberger, von Gudenberg 2002]. The algorithm is based on the infamous Sugiyama algorithm [Sugiyama, Tagawa, Toda 1981]. The authors expand on Seemann's extended version [Seemann 1997] of Sugiyama algorithm. The algorithm follows an incremental layout strategy and is a combination of hierarchical layout and orthogonal layout methods. It implements ranking (layer assignment), minimizes edge crossing and positions classes and two types of edges mainly hierarchical and non-

hierarchical ones. The hierarchical relations such as inheritance hierarchies are drawn using the hierarchical layout with the rest drawn using orthogonal methods. Based on this initial work the author then presents a set of fourteen aesthetic criteria [Eichelberger 2002] many of which are borrowed from graph aesthetics and adapted to the class diagram. This algorithm has been continually updated to include all the aesthetic criteria presented in [Eichelberger 2002]. It is implemented in a Java framework named *SugiBib* [Eichelberger 2001; Eichelberger, von Gudenberg 2003a].

An overview of current UML class diagram tool implementations generally shows a lack of useful UML layouts [Eichelberger, von Gudenberg 2003b]. In this work Eichelberger et al. call for standards in UML class diagram layout in order to make the reading and comprehension of class diagrams easier. The hypothesis is that laying out class diagrams while conforming to aesthetics improves the readability of class diagrams.

Eichelberger also investigated the effect of object-oriented design, cognitive psychology and human computer interactions (HCI) on the UML aesthetics criteria [Eichelberger 2002] for class diagrams [Eichelberger 2003]. Based on object-oriented design criteria, the author suggests incorporating information such as annotated complexity stereotypes, spatial distribution, scaling according to complexity and coloring into the set of aesthetics criteria for the layout of a UML class diagram. These are general design guidelines. Based on the criteria presented, a set of metrics can be derived to determine the quality of a layout. No quantitative evidence has been shown for any of the proposed aesthetic criteria presented in *SugiBib*. A detailed description of all layout metrics are described in Eichelberger's Ph.D. thesis [Eichelberger 2005].

Recently, Eichelberger et al. [Eichelberger, Schmid 2009] presented visual guidelines (based on their previous work stated above) for the aesthetic quality of UML class diagrams as a framework to improve the quality of UML class diagrams. To validate this, they describe a pilot study to determine the effect of the guideline rules on comprehension. This is the first recent pilot user study besides the controlled experiments conducted in this dissertation that attempts to validate UML class diagram guidelines. Eighteen students participated in this study. The experiment was repeated six times in a different domain, each time with the same subjects all in one sitting. A set of diagrams consisting of 13 class-like elements and 12 relationships were drawn. The base diagram followed all the visual rules, whereas the five modified diagrams violated exactly one of the following rules: clear structure in hierarchy (S1), median position of elements in hierarchies (S3), centering n-ary associations (S4), center association classes (S5), and keeping comments close to related elements (S7). Five tasks were given for each of the six domains. The dependent variables are time and number of faults (accuracy) found. Results indicate that the analyzed layout rules have a small effect compared with other characteristics of various diagrams used. In other words, their results did not support the use of the layout rules. There are three main things that differentiate our work from this study. First, they do not consider the semantic closeness of classes and focus mainly on UML notation variants. Second, the tasks presented in our studies are more fine-grained software maintenance tasks whereas the tasks in their study are related to UML notation such as changing the relationships between existing classes or naming the classes derived from a parent class. Third, the diagrams in our

study are representative of real systems since they are based on reverse-engineered designs. The diagrams used in [Eichelberger, Schmid 2009] were not based on real systems and included very few attributes and method names. We consider their approach to be simplistic in that it is not representative of real-world applications. The same author also proposed some guidelines for the automatic layout for UML use case diagrams, another type of diagram to document user stories, and states that the rules needed for layout are much smaller than class diagrams [Eichelberger 2008]. A prototype is shown using *SugiBib*, their layout framework.

von Gudenberg et al. propose an evolutionary layout algorithm for UML class diagrams [von Gudenberg, Niederle, Ebner, Eichelberger 2006]. The layout evolves the position of classes and relationships. In particular, inheritance and associations are evolved. The fitness function is computed based on a combination of nine metrics measuring aesthetic criteria such as edge crossings and number of bends. Mutation operations such as move node, move edge, move bend, delete bend, move port of an edge and flip port of an edge are used with a specified probability. This algorithm runs twenty times slower than *SugiBib*.

Eiglsperger et al. [Eiglsperger, Kaufmann, Siebenhaller 2003] present a new topology-shape-metrics automatic layout method for UML class diagrams based on graph aesthetic criteria. The aesthetic criteria used in this algorithm in priority order are minimize edge crossings, bends, edge length, area, flow direction, hyper-edges (joined inheritance edges), and orthogonal edges. Symmetry and centering association nodes are not explicitly handled. They argue that current hierarchical approaches like the

Sugiyama algorithm work well with class diagrams that have many inheritance hierarchies with deep nesting levels. The algorithm they propose works well for class diagrams with different degrees of structural information. The layout produced by their algorithm is easier to comprehend compared to the hierarchical approach. An upward planarization approach [Eiglsperger, Eppinger, Kaufmann 2003] is used. They tested this approach on general graphs and showed that it produces fewer crossings than the hierarchical approach for low height graphs. They do not apply the hierarchical algorithm directly to class diagram layout since the hierarchical method always optimizes the flow criterion first followed by the crossings and bends criteria. However, only generalizations benefit from the flow criteria whereas other relationships do not. The algorithm was implemented using the *yFiles* Java based library [Wiese, Eiglsperger, Kaufmann 2001] for the automatic layout of many types of graphs. The library includes several layout algorithms including layering, tree, force-directed, circular-radial, organic, and orthogonal layouts. The orthogonal layout is used for UML class diagrams. The *yFiles* system is now commercial with academic and evaluation licenses. It works well to generate an orthogonal layout, however semantic information about classes is not taken into account while drawing of the diagram. For example, in order to minimize edge crossings, two classes participating in an association relationship are placed far apart at two ends of the diagram, with a long line connecting them.

Gutwenger et al. [Gutwenger et al. 2003b] propose a new algorithm realized in the *GoVisual* C++ class library [Gutwenger et al. 2003a]. The algorithm balances the following aesthetic criteria; minimize crossings, minimize bends, uniform direction of

arcs in a class hierarchy, no hierarchy nesting, orthogonal edges, merging multiple inheritance edges and labeling edges. First, a mixed upward planarization is computed followed by orthogonal computation. The *GoVisual* framework follows in the footsteps of the topology-shape-metrics approach described in [Eiglsperger, Kaufmann, Siebenhaller 2003]. However, the planarization and orthogonalization step are done differently in the GoVisual algorithm. GoVisual is available as a plug-in to commercial UML tools such as Borland Together and Microsoft Visio. An earlier version of this algorithm is given in [Gutwenger et al. 2001] [Gutwenger et al. 2003a]. Eiglsperger et al. [Eiglsperger et al. 2004] describes both of the above algorithms ([Gutwenger et al. 2003b] and [Eiglsperger, Kaufmann, Siebenhaller 2003]) as complementary alternatives to the Sugiyama hierarchical layout algorithm.

Another approach towards guidelines for class diagrams was proposed by Sun et al [Sun, Wong 2005]. They propose key graph layout criteria, emphasizing UML class diagrams, based on laws of perceptual theories. From Gestalt laws, they state important factors of perceptual organization such as goodness, similarity, continuation, proximity, familiarity, and connectedness. Based on perceptual segregation, they found symmetry, orientation, and contours to be important factors in recognizing an object. They then present some criteria for each of these laws. All together there were fourteen criteria listed for UML diagrams. The law of similarity suggests the use of colors. The law of continuation suggests the minimization of edge crossings and bends. Most of these criteria are common sense ones borrowed from graph aesthetics. They analyze the application of the fourteen proposed criteria using two commercial UML modeling tools:

Rose and *Together* to improve the readability of class diagrams. The subject diagrams were a small Thermometer application and 28 classes from *JUnit*. Both *Rose* and *Together* comply with most but not all of the layout criteria with *Together* performing better than *Rose*. They do not mention a priority list for layout criteria and they do not take semantics of the class diagram in account.

Dwyer [Dwyer 2001] describes a 3D approach for UML class diagram layout based on the force-directed algorithm. These ideas were implemented in their tool called *Wilma*. A usability study was then conducted to determine the effectiveness of *Wilma*. They hypothesize that *Wilma* is useful in communicating class models to users, assists system architects when creating UML models and helps to suggest meta-structure in models to users. These hypotheses were tested qualitatively. A number of expert system architects were interviewed while using the system in a series of planned tasks. Some of the tasks were timed while other required the subject to verbally state their preferences. They found support for their hypothesis based on the correctness of subjects' responses to concrete tasks as well as the verbal descriptions of the tool. This was the only earlier work that attempted to validate their approach using human subjects.

2.2.3 Colors

In 1999, Peter Coad introduced the concept of using colors to signify archetypal behavior for UML classes in a domain (or business) model [Coad, Lefebvre, De Luca]. Four interconnected archetypes were defined along with a color to represent them. The four archetypes are,

- Moment or interval archetype represented by the color pink,

- Role archetype shown in yellow
- Catalog entry like description archetype represented in blue
- Party, place or thing archetype shown in green.

These four archetypes form a domain neutral building block for better object oriented design. The moment interval archetype represents something that occurs at a certain time or over an interval of time. The role archetype is the way a party, person or thing participates. The description archetype is a collection of values that apply repeatedly. A party, place or thing is someone or something playing different roles.

Classes are categorized under these four archetypes using color. Text annotations using the stereotype tag can be used to label the archetype the class represents. This work does not deal with UML layout but it does deal with colors to make the understanding of a UML diagram easier. Archetypes have nothing to do with stereotypes or architectural importance. The archetypes presented by Coad can be used to determine architectural importance based on some criteria. We see this approach as complementary to our method of determining architectural importance via entity, boundary and control class stereotypes. Both our approach and Coad's approach are domain independent.

Evitts [Evitts 2000] presents patterns of style that contribute to the graphical quality of UML diagrams. The use of different fonts, line styles, colors and shading is suggested.

2.2.4 Metrics, Views, and Areas of Interest on Diagrams

We now describe how metrics are used to improve class diagram comprehension. Lange et al. introduce the tool *MetricViewEvolution* consisting of a set of views [Lange, Wijns, Chaudron 2007] to improve the quality of UML models. The views supported are *MetaView*, *ContextView*, *MetricView*, *UML-City-View*, *Quality Tree View* and *Evolution View*. The *MetricView* is explained in more detail since it is more relevant to improving a class diagram. The other views are abstractions of the UML model and are not directly relevant. The *MetricView* combines the existing UML class diagram layout with visualization of metrics. The metrics are visualized using color, size and/or shape. They may be positioned graphically over classes in a class diagram using bars to represent each metric. Since metrics help manage the quality of architecture and design they use them to improve and measure UML class models. The metrics used are class dynamicity, number of classes per use case, number of use cases per class.

- Class dynamicity/complexity: If the class has a lot of edges emanating from it, the class is assumed to be critical thereby deserving more attention.
- Number of classes per use case: This number tells us if related functionality is spread over many design parts.
- Number of use cases per class: This number represents class cohesiveness.

They also conduct a study [Lange, Chaudron 2007] to determine the effectiveness of these views when used for comprehension. Accuracy and effort is measured. It was a questionnaire based study with 100 graduate students. The results indicate that the new views improve correctness by 4.5% and time needed for comprehension by 20%.

A filter based approach to improving the readability of class diagrams is taken by Kollman et al. [Kollman, Gogolla 2002]. They use metrics to selectively choose coherent regions of class diagrams. The goal is to show cohesiveness between classes. Metrics used include Coupling between Objects(CBO), Number of children (NOC) and other standard object oriented metrics. The UML model is reverse engineered followed by the calculation of metrics between classes. A central class along with the class context (dependent on threshold values) is selected and displayed based on the particular metric chosen. Metric names may be optionally shown using UML stereotype notation. Applications of this technique involve finding potential class candidates for refactoring.

Hammouda et al. present a tool environment where UML models can be customized based on the person, task and learning concern at a time [Hammouda, Guldogan, Koskimies, Systa 2004]. Their goal was to find a technique that supports learning which is based on the class model, is customizable and allows the learner to choose between subjects to be learned and supports a certain sequence in specific learning concerns. This environment is achieved with the help of a comprehension pattern. A role in a comprehension pattern can be associated with a model element. If a user wishes to learn about a specific concern then the model element is bound to the role. The result is a customized class diagram filtered based on the specific concern being learned.

Byelas and Alexandru have worked on the visualization of metrics and areas of interest on software architecture diagrams such as UML diagrams of real systems. In [Byelas, Telea 2006], they present a new technique that minimizes visual clutter for

multiple, overlapping areas of large diagrams. The areas of interest are defined using a software metric. Contours and colors are used to shade areas of the diagram. Classes that are part of a particular contour/shading are determined based on a metric value of the software. In [Byelas, Bondarev, Telea 2006], they present a case study to assess the tool on an industrial toolset. They also demonstrate the use of blending and texturing to render several metrics on top of an area of interest [Byelas, Telea 2008b] [Byelas, Telea 2008a]. These ideas are realized in their *AreaView* tool. They evaluate the approach using 30 students [Byelas, Telea 2009a]. The study was conducted as follows. They first asked all the students to draw contours to represent areas of interest given to them as a list. Next, each subject got the hand drawn contours (randomly and not necessarily their own) and a computer generated diagram by *AreaView*. They had to rank the ease of understanding of the areas, area complexity, rank perceived similarity and explain what they liked least and most in the given drawings. 94% of the students preferred the human-made diagram over the tool-generated one. This led them to add some algorithm improvements to the tool to overcome the drawbacks the subjects talked about. Next, rather than repeating the user study, they conduct a quantitative analysis using a distance map between contours of the two diagrams: human generated and computer generated. This was done for diagrams generated by both the original algorithm and the improved algorithm. Based on this, a distance graph is presented, which shows that the new algorithm is consistently closer to the human drawings than the original algorithm.

In their most recent work, they extend their existing rendering technique to include several metrics including ones with missing values on overlapping areas of

interest [Byelas, Telea 2009b] as well as include metrics on several levels of detail such as classes, methods or groups of classes [Byelas, Telea 2009c].

It is important to note that the above work does not change the layout of the UML class diagram in order to preserve the mental map of software engineers. Keeping the layout constant, related classes with respect to a metric(s) are shown shaded in contours and colors on the diagram. In this dissertation, we propose the changing of the layout for better readability of the diagram. Our work can be seen as complementary to the above work. The above work gives a good overview of the entire system. The work in this dissertation focuses on a specific module and adjusts the layout of classes in that module.

A set of metrics typically used to compare UML class diagrams in the literature is given by Yi et al. [Yi, Wu, Gan 2004]. These metrics measure the complexity of the UML class model featured in the class diagram. Layout of UML models is not taken into consideration.

2.2.5 Focus + Context

Focus + context [Card, Mackinlay, Shneiderman 1999] is an information visualization [Ware 2000] technique that tries to maximize the use of the screen by presenting large amounts of data in a small space. The area of focus is shown in detail while the rest of the space i.e., the context is abstracted. Focus + context is an application of the fish eye view [Furnas 1986] by Furnas.

Köth and Minas use focus + context abstraction mechanisms in their tool *DIAGEN* when dealing with large class diagrams [Köth, Minas 2001]. Abstractions are treated as diagram transformations. Two abstraction transformations or selective

aggregations for simplifying classes and packages are defined. This was a very simplistic application of focus + context for class diagrams.

Musial et al. [Musial, Jacobs 2003] present a focus + context information visualization technique to interact with UML class diagrams. The purpose of this work is to preserve UML notational syntax while improving the space efficiency. A fisheye lens displays less detail for areas with a small degree of interest. Selective aggregation techniques are applied to hide elements (both classes and relationships) that do not fall in the current degree of interest. A level of detail is determined by selective filtering and change in graphical symbols representing the elements. Six levels of detail for classes are supported where level six completely hides the component and level one shows all details. The levels in the middle progressively abstract more information about the class. Classes are displayed according to a degree of interest based on access frequency of the class and its distance from the class in focus. One class is usually taken as the focal point. Selective aggregation of relationships is done by showing only relationship ends. The layout algorithm positions inheritance down the vertical axis as stated by graph aesthetics. Other aesthetics are not taken into account. Since the layout algorithm involves movement of classes and relationship, a smooth animation of the move is chosen to preserve the user's mental map. The author added this interactivity feature to the open source UML editor, ArgoUML.

A focus + context visualization using onion graphs was published by Kagdi and Maletic [Kagdi, Maletic 2007]. The focus of the class diagram is drawn using normal UML notation and is represented by a set of classes instead of just one class. The context

is presented at different levels of detail using onion graph notation. The goal here was to eliminate selective edges in the context of the class diagram. Pure onion notations abstract same type relationships where as mixed onion notations abstract disjoint relationship groups conforming to a set of properties. Selective aggregation is done using sibling-order compaction and level-order compaction. Order of the original relationships and classes in the layout is preserved. The method was illustrated by a series of examples using *Hippodraw* [Kunz 2006] as the subject system. Their approach can be used together with the evidence generated from this dissertation to realize a layout aware focus+context adjustment algorithm.

2.3 Empirical Studies on UML Class Diagram Layouts

This section discusses several empirical studies done with respect to UML class diagram layouts and stereotypes. The first sub-section is dedicated to extensive empirical studies done on UML class diagram syntax, user preference and user comprehension. The next section talks about empirical studies where the use of domain specific stereotypes was the main factor influencing comprehension of class diagrams.

2.3.1 Aesthetics

Purchase conducted substantial empirical work in analyzing aesthetics of graphs including UML class diagrams. She proposed three axes for classifying usability studies: nature of the graph (syntactic vs. semantic), type of usability measurement (preference vs. performance) and the effect being investigated (algorithm vs. aesthetics). A priority list of aesthetics for UML class diagrams was proposed based on her experiments. For

instance, arc crossings and orthogonal edges are considered to be the most important aesthetic criteria for class diagrams. Additionally, the author points out that we need to consider the ultimate use of a graph in order to produce a useful visualization of semantic information. The experiments conducted are now described.

In this study, Purchase et al. [Purchase, Alder, Carrington 2000; 2002] investigate semantic user preferences for UML class and UML collaboration diagrams. The goal of this experiment was to present an ordered list of important aesthetic criteria preferred by users when using UML class and collaboration diagrams. They evaluated six aesthetics namely, minimizing bends, minimizing edge crossings, orthogonal edges, layout width, text direction and type of font used. For UML class diagrams, two additional aesthetics involving relationships were used namely, joined inheritance edges and directional indicator labeling. For UML collaboration diagrams, adjacent arrows and arrow lengths were used as the two additional aesthetics.

One class diagram and one collaboration diagram were used in the experiment. The class diagram had fourteen classes and eighteen relationships. The collaboration diagram had twelve objects and seventeen messages. Each diagram was hand drawn twice for each of the six aesthetics. Subjects were simply presented with a set of diagrams and had to decide which one they would prefer to use in a software engineering task. They were also asked to rank the top three and bottom three diagrams of their choice. This experiment did not take performance with respect to a particular task into account. They came up with the following ordered list of aesthetic criteria for both UML class and collaboration diagrams; edge crossings, orthogonal edges, information flow,

bends, text direction, layout width and font type. For class diagrams, subjects preferred joined inheritance arcs and directional indicators. Adjacent arrows were not preferred in UML collaboration diagrams. The results indicate that second to minimizing crossings, orthogonal edges were preferred by subjects in UML diagrams. This shows that algorithms designed for abstract graphs (where orthogonal edges did not play a major role) do not necessarily work for specific domains of graphs like UML diagrams. A similar experiment [Purchase, Carrington, Alder 2000] was conducted with comparable results.

In another study, Purchase et al. [Purchase et al. 2001] conducted a user performance experiment to determine the comprehensibility of UML class diagram notational variants. This study was not based on user preference. This experiment considered five notations for UML class diagrams each of which had two semantically equivalent, syntactically different variants. The variants were based on the way relationships were presented syntactically. They were related to inheritance direction, inheritance arcs, association representation, association names, and cardinalities. The goal of the study was to determine which variant is more suitable for human performance. Subjects were asked to match a given text specification with a set of experimental diagrams. There were thirty-four novices and five experts. Speed and accuracy were measured. Experts preferred inheritance arrows upwards, joined inheritance lines, association names near classes versus as a separate class. They state that the best notation to use will ultimately depend on the tasks for which the diagram is used. They found the less intuitive ones to help more since it induced them to work

harder in the analysis increasing performance. However, Nielson et al. [Nielsen, Levy 1994] suggest a strong correlation between user preference and performance.

Another series of experiments was conducted by Purchase et al. [Purchase, Alder, Carrington 2002] to determine the effect of aesthetics criteria on user preferences in UML class diagrams. They compared several pairs of class diagrams, where each pair conformed to certain aesthetic criteria presented in a different way graphically. They gathered quantitative and qualitative data based on these pairs of class diagrams. The results show the most important aesthetic criteria for UML class diagrams. Joined inheritance arcs and directional indicators were among them.

In yet another study, Purchase et al. [Purchase, Carrington, Alder 2002; Purchase, McGill, Colpoys, Carrington 2001] try to identify the most important aesthetics in UML class diagram layout from a comprehension perspective. The goal was to determine the diagram that gave the best performance. They conducted two experiments. In the first experiment they used five graph drawing aesthetics namely; minimize bends, even node distribution, uniform edge lengths, consistent direction of edge flow and orthogonal edges. This experiment measured performance quantitatively via speed and accuracy of responses. In the second experiment, they added two other criteria, edge lengths and symmetry. The second experiment only collected preference information. Thirty subjects took part in this experiment. Unfortunately, this experiment's results were inconclusive. The paper indicates that even though the results are inconclusive, it does not necessarily mean that layout does not play a role in comprehension. They state that a more semantic

grouping of nodes might be needed. This is exactly what this dissertation research is addressing using architectural importance of classes.

2.3.2 Stereotype Usage

Kuzniarz et al. [Kuzniarz, Staron, Wohlin 2004] conducted a preliminary study investigating the role and effect of stereotypes in UML class and collaboration diagrams within the telecommunication domain. The main question this work addresses is,

- Does the introduction of stereotypes in UML models influence model understanding?

The goal of the experiment was to analyze the comprehension of UML models in order to evaluate UML stereotypes. The stereotypes used were telecommunication industry specific and were represented using graphical icons to determine if they would help in UML model comprehension. The experiment used two models A and B each with a stereotyped and non-stereotyped version. A description of the stereotypes was also included with the stereotyped model. Forty-four subjects were randomly assigned to two groups. The subjects were given the same questionnaire consisting of twelve questions for each model (stereotyped and non-stereotyped). The questions asked how many classes were present of a specific type, number of different types of elements and checking for accurate element placement based on their definition. Subjects were asked to record the time they took to complete the questionnaire. They collected accuracy and timing information for the questionnaire. The time taken to answer questions was shorter for the stereotyped models with higher accuracy. The results of this study statistically

prove that the use of stereotypes helped in system comprehension. An extended version of this work is given in [Staron, Kuzniarz, Wohlin 2006].

In another study, Staron et al. [Staron, Kuzniarz, Thurn 2005] empirically assess the use of stereotypes to determine improvement in reading techniques for inspecting class and collaboration diagrams. The following question is being addressed,

- Do stereotypes influence the effectiveness and efficiency of finding faults?

In this study the authors evaluate how the use of stereotypes influence the result of three reading techniques used for UML model verification and validation. The reading techniques used were checklist based reading, perspective based reading and unstructured/ad-hoc reading. The reading techniques were not used as an independent variable due to the small subject sample. Eleven subjects were presented with the UML class and collaboration diagrams (both stereotyped and non-stereotyped) as well as the requirements specification in plain text. They were given fault report forms, check list for the checklist based reading and description of the designer's perspective for the perspective based reading technique. The stereotypes used were telecommunication industry. They measured the inspection time and number of faults. The results of this study show that using stereotypes gives 76% improvement in efficiency and 17% improvement in effectiveness of reading techniques. They found that efficiency and effectiveness were positively influenced by adding stereotypes for all three reading techniques. The checklist based reading technique was found to be the most efficient and the perspective based reading technique was found to be the most effective for

investigating faults. These results are only preliminary and need further testing to be generalized.

Ricca et al. [Ricca et al. 2006; 2007] conducted a series of three experiments based on UML stereotyped diagrams and their effect on comprehension on two web applications. In all the experiments they compare two forms of design diagrams, standard UML diagrams and UML diagrams extended with Conallen's stereotypes [Conallen 2003]. The main questions investigated in the series of experiments are,

- Do UML diagrams extended with Conallen's web specific stereotypes contribute to the comprehension of web applications?
- Does subject ability and experience support the use of stereotypes in UML diagrams?

Conallen provides the Web Application and Extension (WAE) stereotypes for UML that support the navigation structure and dynamic page generation typical to a web application. They also have a straightforward mapping to the implementation. The diagrams both UML and UML with Conallen stereotypes were reverse engineered from code and manually laid out so that the diagrams represented a meaningful and good abstraction of the code. They also used source code in the experiment. They measure the comprehension level of subjects. Before the experiments subjects had been trained on Conallen's notation and performed a simple comprehension task on a test application.

Results of the first experiment were reported in [Ricca et al. 2006]. Subjects were 35 undergraduate students. The students were supposed to understand part of two web applications. The web applications used were *Claros* and *WfMS*. The number of classes

used were 78 and 92 for each application respectively. Students were randomly assigned to four groups. They measured comprehension by asking 12 open questions. Real change requests from web applications in SourceForge [SF] were used to devise the questions. Students were allowed to look at UML diagrams, source code and even use the web application. The answers to each question were expressed as a list of classes or webpages. The precision, recall and F-measure were used to measure comprehension level based on the answer list. The results of the experiment statistically show that Conallen's stereotypes significantly support web application understanding.

Results of the next two experiments were published in Ricca et al. [Ricca et al. 2007]. They also examine the effect of experience and ability in subject's comprehension level in UML diagrams while using the same Conallen's stereotypes as in their previous experiment [Ricca et al. 2006]. The results also take into account the study results obtained from [Ricca et al. 2006]. The main question being investigated was the same as stated above since the previous experiment was part of a series of experiments done over time. The subjects in addition to the 35 undergraduate students of the above experiment included 13 graduate students at one university and another 18 graduate students at another university. The ability level (low, high) was ranked according to their performance in courses taken at the university. The experience level was determined by graduate vs. undergraduate students. They formulate the following hypotheses

1. When performing a comprehension task the use of stereotyped class diagrams vs. non-stereotyped class diagrams significantly improves the comprehension level achieved by maintainers.

2. Subjects' experience and ability significantly interact with the use of stereotyped class diagrams to influence the comprehension level achieved by maintainers.

The results show that stereotype usage was not the only factor that influenced comprehension performance. The experiments that involved graduate students found that Conallen's stereotypes did not significantly help in comprehension. It helped only in the experiment that involved undergraduate students. In conclusion, they found that high ability graduate students on average performed better using standard UML class diagrams while low ability undergraduate students perform better using Conallen's stereotypes. Using Conallen's stereotypes thus reduces the mean difference between high and low ability subjects as well as between graduate and undergraduate subjects. Thus the first hypothesis mentioned above is rejected and the second one was accepted based on statistical results. They also found subjects spent more time on stereotyped diagrams but this did not lead to a significant improvement of software comprehension. Experts also preferred to look at the code instead of diagrams. However an important observation made was that using Conallen's stereotypes helped to reduce the difference between experienced and inexperienced users. A detailed description of the above studies by Ricca et al. are given in [Ricca et al. 2010].

2.4 Eye-tracking Studies

Eye trackers have evolved significantly over the past decade making them easier to use for study participants and experimenters. Study participants are no longer constrained to head mounting devices making them non-intrusive. Experimenters benefit from ease of conducting the experiment and software available making it easier to

analyze eye tracking data and derive conclusions. The result is an increase in the amount of studies published using eye trackers. State-of-the-art eye trackers accurately determine eye movements with a short calibration time needed before the study is conducted. A number of studies are conducted using eye trackers in different areas of research. However, eye tracking based experiments on software artifacts are not very common. This is expected to change in the near future as eye trackers continue to become increasingly affordable. The following sub-sections discuss eye tracking studies done in domains other than software engineering as well as a few recent ones done on software artifacts. It can be seen that software engineers have only scratched the surface when it comes to the usage of eye trackers to assess processes and tools.

We now give some eye tracking [Duchowski 2003; Rayner 1998] terminology that will be used when discussing eye tracking related studies. A *fixation* is the time duration for which the eyes focus on a point of interest. *Saccades* are rapid eye movements that occur between fixations. A *scan path* is a directed sequence of fixations. Theoretically eye fixation data is linked to comprehension [Bouman 1973; Just, Carpenter 1980].

2.4.1 Usability Studies

Majority of work using eye trackers falls under the rubric of usability studies. Bojko presents the process involved in designing and conducting good studies involving eye tracking based usability testing [Bojko 2005]. The paper presents a set of questions eye tracking may provide an answer for along with a set of guidelines. Cowen [Cowen 2001] gives a detailed account of eye movement analysis of the usability of web pages.

2.4.1.1 Web Sites

We first present eye tracking studies related to web sites. Cutrell et al.[Cutrell, Guan 2007] conducted an eye tracking study to explore the effects of changes in the presentation of search results given by search engines. They investigate how users read descriptions/snippets of search results and if scanning search results are different for navigational versus informational tasks i.e., the context and type of the search. The experiment with three search snippet lengths with twenty two subjects was used to determine if this affects the performance of users. They found that people viewed results in a linear order with most attention given to the first few items. They also found that changing the snippet length had no effect on the search behavior. The results also show that adding information to search query result snippets improves the information tasks but degrades navigational task performance.

Beymer et al.[Beymer, Russell 2005] develop the *WebGazeAnalyzer* system which captures and analyzes detailed eye movements of users reading web pages and performing web-based tasks. The method of capturing and analyzing eye gaze data is described. A typical study involving websites measures how many times a user visits a page, duration of the visit and eye gaze patterns on the page. This tool captures the entire web session in a movie, the parsed DOM of each web page, eye gazes and window events. It computes groups of fixations corresponding to lines, time spent reading, what content was being read, reading speed and regressions. A regression occurs when a user goes back and fixates on something they just finished fixating on. They deal with noise and drifts found in remote eye trackers. The main contribution of this work was the

building of the tool and handling drift error in the eye tracker. They used the Tobii 1750 eye tracker. Reeder et al. [Reeder, Pirolli, Card 2001] uses eye gaze data along with the HTML DOM to find eye gazes to an HTML element using their tool, *Web-EyeMapper*.

Goldberg [Goldberg et al. 2002] conducted an eye tracking study to assess design features of web applications. They tried to relate eye tracking parameters to page sequence or user actions and whether users looked at items horizontally or vertically. Tasks involved editing and maintenance of links on the web portal. They also studied fixation order within defined regions of a screen. The results indicate that important items should be placed on the left and top of the web portal to minimize search time.

A study was conducted by Nakamichi et al. to detect low usability of web pages using user behavior including eye movement [Nakamichi, Shima, Sakai, Matsumoto 2006]. They used mouse movements and browsing time to determine this. The moving gaze speed helped in deciding low usability pages. 94.4% of the low usability web pages were detected using just the moving gaze speed and mouse wheel rolling. Low usability pages had a higher browsing time, higher mouse move distances, higher mouse wheel rolling, and higher moving gaze speed with low mouse speed.

In another eye-tracking study, Pan et al. [Pan et al. 2004] explore the factors that affect web pages. In particular, the gender, type of web sites, order of web pages being viewed and the intended tasks were analyzed. The eye tracking metrics used were mean fixation duration, gaze time and saccade rate. The results point out that web page viewing is affected by gender, order of pages and relationships between site types, and order of pages.

Whalen et al. conduct a study to determine how users view security cues in web browsers [Whalen, Inkpen 2005]. Eye tracking and questionnaires were used to collect data. The questionnaires were used to check if the users' answers match the way they actually behaved in terms of their eye movements. They did find discrepancies where the users thought they clicked something but in fact eye tracking data showed that they had not. The results indicate that people usually look at the lock icon but do not use it interactively and use the certificate information very rarely. They also found that once a user logs into a site they stop looking for security features. Their end goal was to inform users of security mechanisms on websites without unnecessarily distracting them.

The micro behavior of users reading web pages and doing web related tasks was measured by Card et al [Card et al. 2001]. They use information foraging theory [Pirolli, Card 1999] to develop methods for analyzing a web task using the information scent of a link. They develop models of how people explore the web using eye tracking and logging mechanisms.

A study on browsing multiple search pages is presented in [Matsuda, Uwano, Ohira, Matsumoto 2009]. They analyze eye movements during web search and the amount of time spent viewing each result. Results show that search results displayed on the top of the latter page were viewed for a longer time than those displayed on the bottom of the former page.

Two usability evaluation methods: heuristic evaluation and empirical user testing using eye-trackers are compared by de Kock et al. [de Kock, van Biljon, Pretorius 2009] based on a website for a learning management system. Their purpose was to investigate

the differences in usability information. Results indicate both are useful approaches, having different goals and also differences in the type of information rendered. For example. The heuristic approach deals with the *why* and *when* and the eye-tracking approach deals with the *what* and *how*.

2.4.1.2 Computer Based Tasks

We now present eye tracking studies based on everyday computer based tasks. Iqbal et al. [Iqbal, Zheng, Bailey 2004] conducted a study to determine the effect different types of daily computer based tasks have on the mental workload. The tasks involved were daily compute related tasks a user typically performs. Since pupil size has been shown to correlate with mental workload for non interactive tasks, they used the pupillary response with respect to interactive tasks as a measure of mental workload. They used a head mounted eye tracker to measure pupil size. The questions they investigated can be stated as follows,

- Does the pupil size reflect mental workload within a category of tasks?

The independent variables being studied i.e., were the task categories namely, object manipulation, reading comprehension, mathematical reasoning and searching. Each of these task categories had an easy and difficult level. In total, each user performed eight tasks. In addition to pupil size, they also measured task completion time and users' subjective ratings. A hierarchical task has a varying level of mental workload and pupillary response correlates with the changes in workload. A task is considered hierarchical if it can be decomposed into lower level tasks. The results indicate difficult

tasks require longer processing with higher mental workload accompanied by greater pupillary response.

As an extension to this work, Iqbal et al. also illustrated changes in mental workload during different types of tasks namely, interactive and hierarchical [Iqbal, Adamczyk, Zheng, Bailey 2005]. The goal of this study was to better understand how the workload changes and by how much does it change at subtask boundaries. This study mainly investigated the following questions,

- How much change is detected in the mental workload during subtask execution and at subtask boundaries? Does the change depend on the level and type of the subtask?
- How can one map the mental workload to a computational index for better reasoning?

The study involved two tasks, one was a route planning task and the second tasks dealt with editing a document. Each task had subtasks with varying levels of difficulty. The results indicate that different types of subtasks incur different workloads on a user. The workload decreases at subtask boundaries. Since the workload decreases, this would be the ideal time for any interruptions to the user causing less disruption. They also found that mental workload decreases more at the boundaries higher in the task model and less at the boundaries lower in the task model. Mental workload also changes within subtask boundaries at the same level of a task. The index of opportunity maps pupillary response to a twenty point scale. The scale indicates the time that is ideal for an interruption to occur, where one indicates the least ideal time and twenty indicates the

best time to interrupt a user. This work was the first showing evidence of changes in mental workload at different levels and boundaries of tasks.

2.4.1.3 Eye Trackers as Input

Jacob [Jacob 1990; 1995] discusses several factors, technical and human related, that influence the usefulness of eye movements as input in user interfaces. This work is mainly interested in using eye movements to interact with the computer via a dialogue in real time. This has applications in fields where the user's hands are occupied with the mouse and keyboard but may be able to use eye movements as an easier method of input. Murata [Murata 2006] also looks into eye gaze input versus the mouse to support reduced motor functions of the elderly.

The relationship between user understanding and eye movements is studied in [Khat, Matsumoto, Ogasawara 2004]. The purpose of this study was to discover whether a user found it difficult to comprehend when reading text in a non-native language. The system proposes a translation of the word when this is detected. The results indicate that regressions can be used to trigger help processes.

Pirolli et al. [Pirolli, Card, van der Wege 2003] contrasted the use of the hyperbolic tree browser with a file browser. An eye tracker was used for data collection. The hyperbolic tree is a focus+context technique that is used to navigate large tree structures. Information scent is a technique that provides cues to a user such as a label near a node in the tree that influences the search for a particular node. There was no difference in the performance time between the two browsers. However, more tree nodes were examined using the hyperbolic tree and search was conducted faster than the file

browser. They found information scent to improve the visual search. When information scent was high, the hyperbolic tree browser was faster but slower when the scent was low. More fixations were noticed with the hyperbolic tree but the durations were shorter. The hyperbolic tree let users search more of the tree at a faster rate.

The cognitive strategies used to search for an item in a hierarchical display such as a menu with category headings is discussed by Hornof and Halverson [Hornof, Halverson 2003]. They found that organizing data in a hierarchical layout was easier to search than non-hierarchical layouts. In addition to this, the results provide a starting point for screen layout guidelines.

Vonder Embse conducted an eye tracking study on mathematical graphs [Vonder Embse 1987]. The results indicate that there was a significant difference in the way experts and novices read graphs. Experts used a small number of lengthy fixations whereas novices used a large number of short fixations. This shows that experts can easily focus on key areas of a complex mathematical graph whereas novices cannot. They also found that in general the duration of fixations on graphs is longer than for regular text.

2.4.2 Software Engineering Studies

We now discuss eye tracking studies within the software engineering community. To date, there are only a few researchers who have used eye tracking measures in studying software artifacts. They are categorized in the following sub-sections.

2.4.2.1 Source Code Comprehension

In one of the earliest eye tracking studies, Crosby et al. [Crosby, Stelovsky 1990] discover the reading methods programmers use when viewing an algorithm implemented in code. Crosby and Stelovsky [Crosby, Stelovsky 1990] study eye gaze data of novices and experts to determine if experience had an effect on viewing patterns. The main questions being addressed in this study are as follows,

- Is there a lag between viewing and processing source code (complex text)? Does the immediacy theory hold?
- Does programming experience alter algorithm reading strategies affecting how or where we look?
 - Does programming experience influence our preference for comments or code?
 - Does programming experience influence whether focus is on critical areas of the code implementing the algorithm?
- Do we read algorithms the same way we read text or does the strategy change to accommodate complexity?
- Does text in graphical representations of algorithm behavior used to support the understanding of an algorithm draw as much attention as the graphics?

The experiment was conducted on a short algorithm namely, the binary search written in the Pascal programming language. Eye scan patterns, fixation durations and number of fixations spent reading were recorded. Nineteen subjects who participated in

the study were rated at a low level (ten subjects) and high level (nine subjects) of experience with regards to programming. The binary search was demonstrated on two input sets of fifteen and a hundred-and-twenty numbers each. The graphical representation consisted on a bar chart showing how the search progressed.

The results show that the immediacy theory holds, which means that the numbers of fixations were actually things the subjects devoted attention to. Low experienced subjects tended to spend more time reading comments than high experience subjects. The high experience subjects paid more attention to complex statements in the code. Both low and high experience subjects did not display difference in their reading strategies. Keywords in the programming language were the looked at the least even though these are usually the ones highlighted in source code editors. Results related to the use of graphical representations were inconclusive. They also found that viewing strategies for algorithms are different from viewing strategies used in prose.

Uwano et al. [Uwano, Nakamura, Monden, Matsumoto 2006] study eye viewing patterns of five subjects while they detect defects in source code. Several eye tracking measures are used to determine performance in source code reviewing. The experiment analyzed thirty source code reviews. Source code is not read in the same way as text documents. Frequent jumps are made between lines. They developed a mechanism to measure eye movements and compute the line number of source code that a reviewer is looking at. This was realized in their tool *Crescent*. They identified a specific scan pattern in eye movements. The scan pattern is characterized by the reading of the entire code before going into the tiny details by line. 72.8% of lines of code were looked at in

the first 30% of time. Reviewers who did not spend time during the scan pattern took longer to find defects during the review. Their eventual goal is to help a reviewer find the most defects in programs. Their recent work focuses on multi-document review [Uwano, Monden, Matsumoto 2008]. The system records eye gaze data and mouse/keyboard input for analysis. They calculated the defect detection ratio, detection time per defect and fixation ratio of eye movements on each document. Results indicate that reviewers who concentrated their eye movements on the requirements document found more defects in the design document, which suggests that developers read high-level documents when reviewing low-level documents. A methodology to track eye gaze data for a pair of programmers simultaneously is presented in [Pietinen et al. 2008].

Aschwanden and Crosby [Aschwanden, Crosby 2006] conducted an eye tracking experiment to study how programmers read code. Since beacons have shown to aid in program comprehension, their hypothesis is that beacons were used more by expert programmers compared to novices. A beacon is a chunk of code that aids in program comprehension. The questions they were trying to address can be stated as follows,

- How do programmers read and understand source code?
- Is there a difference between programmers who have different levels of experience?

The experiment comprises of six simple algorithms in recursive and non recursive form for a total of twelve algorithms. The algorithms used were Sum, Exponent, Factorial, Binary Search, GCD and Fibonacci. Each was accompanied by a series of questions. They found that more time was spent on correct answers. Everyone had

different reading speeds. It was found that some lines were given more attention determined by the average fixation duration and longest fixation. They did not find any relation between answers (correct or incorrect) and scanning behaviors. No relation between number of programming years and correctness of answers was reported either. Their results suggest that long fixation durations of 1000ms or greater should be considered to be beacons however statistical analyses did not prove this.

2.4.2.2 UML Class Diagrams

Three eye-tracking studies are closely related to the research presented in this dissertation. These are described next.

Guehénéuc [Guéhéneuc 2006] used a head mounted eye tracking system to investigate the comprehension of UML class diagrams. They use a visualization technique to present data collected from the eye tracker. They implement this in their tool named *TAUPE*. The research question they addressed is stated below.

- How do software engineers obtain design information from a class diagram during program comprehension?

They performed two case studies each with two different class diagrams and program comprehension activities related to them. They use the EyeLink II system for data analyses. The experiment was conducted in a set of trials where a trial consists of one subject looking at a single image. They collected fixations, blinks, and saccades in Eyelink Data Files (EDF). The visualization tool they build aggregated the data from the EDF files and presented it graphically. The visualization mainly showed parts looked at

by software engineers to retrieve information. In order to equate fixation with attention, they identify certain areas of interest before the experiment. Areas of interest include class bounding boxes and measure fixation and saccade aggregation in those areas. Subjects included 12 graduate students who were asked one question for each class diagram. The results indicate that initially the software engineers browse through the class diagram in a random fashion to identify most useful parts. They then focus on parts related to the question asked. They found that software engineers do not use relationships such as inheritance, dependency, association, aggregation and composition which was a surprisingly result. This was because of the nature of the questions and the simplicity of the class diagrams used.

Yusuf et al. [Yusuf, Kagdi, Maletic 2007] conducted an eye-tracking study to assess the comprehension of UML class diagrams using a state-of-the-art eye tracker. The use of layout, color, and class stereotypes (control, boundary and entity) were all assessed to determine their effectiveness in program comprehension. The results indicate variation of eye movements between experts and novices in both UML expertise and software design ability. A detailed description of this study along with results is given in Section 5.1, since it is an extension to the pilot study presented in Chapter 3 [Andriyevska, Dragan, Simoes, Maletic 2005].

Recently, Jeanmart et al. [Jeanmart, Guéhéneuc, Sahraoui, Habra 2009] conducted a study on the effect of the Visitor design pattern on program comprehension and maintenance using an eye tracker. The study considers two types of tasks: comprehension and modification. Three design alternatives were used: diagrams with no

patterns, diagrams with the canonical pattern layout and diagrams with the pattern in a modified layout. They used the average duration of relevant fixations and non-relevant fixations and the normalized rate of relevant fixations as the dependent variables with respect to eye gaze data. Twenty-four subjects participated in this experiment. They were placed into three balanced groups. Each group was given a different combination of the three different design alternatives with respect to a different system to avoid learning effects.

One of the results was that the inclusion of the Visitor pattern in a class diagram plays a role in maintenance tasks. No significant difference was found for the comprehension tasks. In particular, the Visitor pattern layout in canonical form as presented in Gamma et al. [Gamma, Helm, Johnson, Vlissides 1995] required less effort from developers. Besides the studies presented in this dissertation, this is the first study to investigate layout in UML class diagrams.

Chapter 7 and Chapter 8 of the dissertation examine the effect of layout on the task of detecting roles in four design patterns using two alternate methods of data collection: questionnaires and eye-tracking equipment. The results state that the multi-cluster layout outperforms the orthogonal layout for the role detection task especially with respect to the time needed. The multi-cluster layout positions classes in the design pattern based on the canonical form presented in Gamma et al. [Gamma, Helm, Johnson, Vlissides 1995]. These results agree with results obtained by Jeanmart et al. [Jeanmart, Guéhéneuc, Sahraoui, Habra 2009] that the canonical form is better than another modified layout.

2.4.2.3 Program Visualizations

A study on the effects of experience in eye gazing during dynamic program animations was done by Bednarik et al. [Bednarik, Myller, Sutinen, Tukiainen 2005]. The tool *Jeliot 3* was used as the program animation/visualization tool for two short Java programs. The hypothesis was that performance and eye gaze behavior of novices and intermediate subjects would be different. The question they were trying to answer is stated as follows,

- Do novices and intermediates look at program animations in the same way?

The experiment recorded eye gaze behavior of participants during program comprehension tasks aided by the program animation. They recorded the relative fixation count over areas of interest, number of switches per minute (between code and animation) and mean fixation duration over areas of interest. Sixteen people (split into novices and intermediates) participated in the study. The Java programs were between fifteen and thirty-eight lines of code. The results show significant differences in performance measures and in fixation durations. Novices took longer to complete the study than intermediates and spent more time on animations. The fixation count and the number of attention switches per minute were the same for novices and intermediates. The mean fixation duration for intermediates was shorter than for novices overall. This is explained by the fact that intermediates form hypotheses quickly while reading the code because of more domain knowledge and programming experience. The mean fixation duration measured the level of processing required.

Bednarik et al. [Bednarik, Tukiainen 2006] state that even though a lot of attention is put into cognitive processes, there is little application of eye movement tracking to aspects of software. They call for more studies due to important behavior that can be revealed using eye-tracking data. Their eventual goal is to build a methodological framework allowing the application of eye tracking to program comprehension and providing a way to analyze data. In the exploratory study presented here, they used eye movement tracking to study comprehension processes of programmers using a visualization tool. The goal of the study was to determine whether the role of different representations of a program changes during the course of time. Sixteen people participated in the study and were unfamiliar with the code. Data from two participants were discarded due to technical problems. The study was conducted on three Java programs. They report results on the time spent on animating the program's execution. They analyzed ratios of fixation counts, attention switching and fixation durations between different representations of a program. Attention switching was measured as number of switches per minute. The program visualization tool has two main areas of interest, the code part and the visualization part. The mean number of animation runs was a good indicator of previous programming experience. Subjects who ran the animation at most twice were experts in programming. Less experienced subjects focused mainly on the visualizations but with subsequent runs focused more on the code. Experts studied the code first before running the visualization.

Nevalainen et al. conducted an experiment visualizing program variables. They investigate how people construct the mental model with textual versus graphical program

variables [Nevalainen, Sajaniemi 2005]. They measured twelve participants' gaze and collected a program summary from them. The two types of visualizations for program variables were done with the *PlanAni* tool for the graphical aspect and the watch feature was used in Turbo Pascal for the textual aspect. They found that people tend to look at program variables more if they were represented graphically. They also found an increase in the amount of high level information about the program with a decrease of low code level information. The same authors conducted another related study but instead of program variables they looked at visualizing operations and determined if this affected program perception [Nevalainen, Sajaniemi 2006]. Examples of operations include assignment statements, comparisons, input, and output. In this study textual cues were preferred to graphical ones. Subjects preferred to browse the code instead of looking at the animations.

2.4.2.4 Program Debugging

The behavior of programmers during debugging activities is investigated in [Romero, Cox, du Boulay, Lutz 2002]. The programmers used a software debugging environment that displayed multiple linked representations of Java source code, a program visualization tool and the output. They used a visual attention tracking system namely, the restricted focus viewer (RFV) [Jansen, Blackwell, Marriott 2003] which is an alternative to eye tracking equipment. The stimulus is presented in blurred form. The study participant moves the mouse to a section of the image after which the area is seen clearly. RFV typically collects mouse and keyboard actions and times to perform the tasks. They address the following questions.

- To what extent do programmers use software representations such as source code, visualization tools, and output?
- Are particular patterns of representational use associated with superior debugging performance?

They conducted a within-subjects experimental design study. They recorded the degree to which programmers used each representation, switches between representations, number of bugs identified, the order in which they were identified and bug discovery latencies to name a few. There were five subjects who participated in the study in three debugging sessions, the first one being only a warm up session. They were asked to find as many errors as they could. In the first phase of the experiment they were acquainted with the general problem. The second phase presented to them three windows containing code, visualization of the program and sample output. The results show that the RFV slowed down participants but they found more errors. Better debugging is found using multiple representations with between 80%-95% of time spent on the use of source code and frequent switches between source code and visualization. They found high frequency switches between code and visualizations related to good debugging performance.

Bednarik et al. [Bednarik, Tukiainen 2004] compared two tools namely the Restricted Focus Viewer (RFV) and a remote eye tracker in debugging three Java programs for ten minutes. They built an environment over the RFV. The code was shown on the left, visualization on the top right and output on the bottom right. Their hypothesis was that the restricted focus viewer changes the strategies and behavior of a

programmer and does not accurately measure all visual attention context switches. The experiment was within-subjects. They recorded errors spotted, accumulated fixation time, mean fixation duration and switching frequency. Eighteen users took part in their study. The results were statistically analyzed using *ANOVA* or paired sample t-tests. They used the remote Tobii 1750 eye tracker at 30Hz. They found that performance in debugging and time spent is almost the same in both tools. However the time spent context switching between areas of interest was significantly different in both tools. The blurring used in the RFV method interferes with the cognitive strategies of programmers. The mean fixation duration was lower with RFV. Hence it was shown that RFV interferes with the natural flow of tasks being analyzed and must be used with caution.

In another study, Bednarik et al. [Bednarik, Tukiainen 2005] tried to determine if the same effect found in [Bednarik, Tukiainen 2004] was found between experts and novices i.e., participants with different levels of experience. In this study their hypothesis was that strategies possessed by experts during debugging are not affected by the tool used. There were eighteen users in this study as well and had never done an eye tracking experiment before, ten of which were novices with low experience. For experts, the overall mean fixation durations and mean fixation duration over the source code significantly differed between the RFV and the eye tracker. Novices did not show any difference. The results indicate that blurring used in RFV interfered with strategies experts use during debugging. It also has an effect on fixation duration. Debugging performance i.e., the number of errors found was the same. These results corroborate with their previous study.

Bednarik et al. [Bednarik, Tukiainen 2008] also investigate debugging behavior of fourteen subjects while they debug a program in an IDE setting. The original study was done by Romero et al. 2003 [Romero, Du Boulay, Lutz, Cox 2003] and later replicated by Bednarik et al. [Bednarik, Tukiainen 2007]. In this study, the authors use data from [Bednarik, Tukiainen 2007] and extend the analysis by segmenting the data into a series of short intervals and conduct a fine grained analysis in time.

2.5 UML Modeling Tools

This section discusses layouts commonly available in commercial tools. Some of the popular modeling tools available are Rational Rose, Borland Together, Microsoft Visio, Magic Draw and Visual Paradigm for UML. A usability study was done using students on two tools, Rational Rose and Visio [Shumba 2005]. From the point of view of diagram output, 82% of the students liked the output from Rose better than Visio since Visio is limited in where the connectors can be placed in a diagram. However, both tools have advantages. Visio was very user friendly and if the only purpose was to produce design diagrams then it was the obvious choice. Rose was the tool of choice if after diagram creation, code generation was also required. Some of the features available in various tools is now described briefly. Visio provides a layout shapes feature that allows changing the placement and connectors in a diagram. Some options for nodes and edge placement include: top to bottom, bottom to top, left to right, right to left, right angle, tree, straight. It is also possible to specify space between classes, the average class size, space between relationships, and space between classes and relationships. The layouts available in Visio are not specific to class diagrams. They are typical graph based layouts

applied as is to class diagrams. Figure 2.1 shows an example of applying a layout to the class diagram given in Figure 1.1. The layout in Figure 1.1 was hand crafted and seemed readable enough, but after applying a Visio layout the diagram is unnecessarily repositioned making it more difficult to read as shown in Figure 2.1. There are also many edge crossings evident in the Visio layout.

Figure 2.1. Visio's layout produced for the sample class diagram in Figure 1. using the *Compact Tree* style for nodes, *Right then down* Direction for edges and *Shallow depth*

Visual Paradigm for UML [VP 2007] is a commercial tool that supports the hierarchical and orthogonal layout. The orthogonal layout is based on the topology-shape-metrics approach. The hierarchical layout is used for inheritance hierarchy based structures. The layouts available in Visual Paradigm are oriented towards class diagrams. It does not merge inheritance hierarchies if the orthogonal layout is selected. There are other layouts based on tree, circular and organic structures.

Some research projects that evolved into commercial applications include GoVisual and yFiles. Both of these tools are based on the algorithms presented by [Eiglsperger et al. 2004] in Section 2.2.2. GoVisual Community Edition supports the hierarchical and orthogonal layout for UML class diagrams. The yFiles system has more sophisticated layout mechanisms including the hierarchical and orthogonal layout for class diagrams. Graphviz is an open source tool [Ellson et al. 2001] tool that targets visualization of abstract graphs however specific layouts for UML class diagrams are not available.

A recent comprehensive survey done by Eichelberger et al. [Eichelberger, Eldogan, Schmid 2009] shows that many tools do not comply with the UML specification. According to the UML compliance definition only four of the sixty-eight tools analyzed were compliant.

2.6 Discussion on Related Work

General graph drawing algorithms work fine for graphs containing simple nodes. A node in a UML class diagram is not as simple. It consists of many parts. A class node

is made up of three parts: the class name, class attributes and class operations. The relative size of the node is much bigger than a general graph structure. Purchase has proven using extensive empirical tests that general graph based guidelines and algorithms do not work well if applied directly to UML class diagrams. Modifications are needed based on the domain of use in this case the UML notation used in software design. Software visualization tools that visualize some code or design structure may also face the same problem if their nodes are not representative of the ones used in abstract graphs; however the problem is less pronounced since they are not required to conform to a standardized notation.

Purchase found that the most important aesthetic preferences for class diagrams were minimizing crossings, orthogonal edges, minimizing bends, horizontal labels and joined inheritance arcs also known as hyper-edges. Eichelberger added class diagram specific criteria to the above such as centering n-ary associations and placing comments and association classes closer to related model elements. Eiglsperger adds to this list some visual perception guides such as drawing generalizations in the same direction, avoiding nesting of hierarchies and using colors to highlight class hierarchies.

Purchase distinguishes between two types of tasks, relational and interpretative, carried out during experiments. Relational tasks measure how a user reads a graph and answers questions about it. They are not dependent on any domain knowledge and are quite generic in nature. Interpretative tasks measure the effectiveness of graphs in a specific application domain. The experiments done by Purchase involve relational tasks.

Our research is interested in both types of tasks with a major emphasis on the interpretative ones.

UML class diagram layout algorithms presented in Section 2.2.2 deal with layout creation which is the process of adding geometric attributes to a graph to create a picture. Layout adjustment is the process of changing the geometric attributes of a graph to adjust the picture [Misue, Eades, Lai, Sugiyama 1995]. This research focuses on the layout adjustment problem. This is viewed as complementary to the algorithms and techniques presented in Section 2.2.2. It can be seen by the diagrams presented in [Eiglsperger et al. 2004] that if an association exists between two classes it is drawn all around the diagram to minimize edge crossings. This is unnecessary and our conjecture is that this inhibits comprehension. There is no quantitative evidence provided to validate the usefulness of these algorithms. Empirical studies such as experiments and case studies are needed in order to determine the usefulness in design and maintenance tasks. Stacy et al. [Stacy, Macmillian 1995] show cognitive bias exists in software engineering tasks and hence empirical investigation is preferred. This research conducts systematic experiments to prove the value of layout techniques.

The metric and view based approaches used for UML class diagrams do not consider layout of the model. They are mainly based on filtering mechanisms. Focus + context approaches also do not consider layout however, they do modify the amount of information presented based on the area of interest. Metrics to determine quality of graphs and UML class diagrams are proposed in the literature such as [Purchase 2002; Yi, Wu, Gan 2004]. These metrics mainly focus on the graph aesthetics or the UML

model represented in a class diagram. In this research, we are interested in determining objective measures to measure the comprehensibility of UML class diagrams taking into consideration layout based on architectural properties.

Some experiments done by Purchase [Purchase, Carrington, Allder 2002; Purchase, McGill, Colpoys, Carrington 2001] to determine the comprehensibility of UML class diagrams had inconclusive results. The researchers suspect that this was due to some other criteria and did not mean that layout was not important. This dissertation directly addresses this issue. The approach taken in this work is to semantically group nodes based on their importance in software architecture. The experiments conducted in this dissertation prove the usefulness of this approach.

A handful of researchers including the author have considered the use of stereotypes to aid in the comprehension of UML class diagrams (see Table 11.1). They do this via controlled experimentation. Besides [Andriyevska, Dragan, Simoes, Maletic 2005] and [Yusuf, Kagdi, Maletic 2007], none of the other researchers consider class diagram layout and stereotype usage together in comprehension of UML class diagrams. More recently, [Jeanmart, Guéhéneuc, Sahraoui, Habra 2009] consider layout in the Visitor design pattern for comprehension and modification tasks. The pilot study [Andriyevska, Dragan, Simoes, Maletic 2005] was one of the first address the symbiotic nature of class diagram layout and stereotype usage.

Stereotypes used by other researchers were specific to a particular domain. Telecommunication stereotypes were used by [Kuzniarz, Staron, Wohlin 2004], [Staron, Kuzniarz, Wohlin 2006] and [Staron, Kuzniarz, Thurn 2005]. Web-specific stereotypes

by Conallen were used by [Ricca et al. 2006; 2007; 2010]. The stereotypes we use are not domain dependent. Every system has control, boundary and entity stereotypes present. Hence our results will not be restricted to any particular domain and can be generalized to object-oriented systems in general. Conallen's stereotypes increases the number of classes displayed in the class diagram which could clutter the layout even further. In [Ricca et al. 2006; 2007], the post experiment survey questionnaire asked users how long they spend browsing code, or if they found the code. These answers are subjected to user bias since a user might not accurately report their behavior. Recording the session would have solved this problem. Also time spent answering questions was not considered.

The other researchers who used stereotypes hand crafted class diagrams viewing the source code to make sure it accurately represents the code. Even though we used handcrafted diagrams, our end goal was to automatically generate an architecturally important layout. The pilot study was conducted to determine the usefulness of the layouts. The software used in the empirical studies using stereotypes are small applications consisting of at most hundred classes, where about twenty classes were actually used in the experiments. All of the studies conducted in this thesis are on real systems. For example, the pilot study presented in Chapter 3 is on a real system, *Hippodraw* consisting of more than two-hundred classes and used a total of fifty classes in our class diagrams. This does not mean that we need to test our hypotheses on large class diagrams but on real systems. Typically this will involve many class diagrams, each explaining cohesive parts of the whole system.

The research aims to improve the cognitive processes [Détienne 2002; Storey 2005; von Mayrhauser, Vans 1996] software engineers use to form a mental model while reading class diagrams. Most of the studies in software engineering used to identify cognitive processes use think aloud protocols and video taping. We consider the use of eye tracking data to help us determine the thought (cognitive) processes that a software engineer goes through while exploring a class diagram. Eye fixations relate to the importance and complexity of information [Just, Carpenter 1980]. Large fixation durations have been shown to increase the cognitive load on a user thereby linked to higher complexity. The number of fixations also determines an element's significance. Other eye tracking measures such as pupil dilation, fixation coverage and saccade length are also available but not used as often.

The Conference on Human Factors in Computer Systems (CHI) organized a workshop that was targeted towards finding a good measure of satisfaction from eye-tracking and defining best practice in eye tracking research. Most of the studies done using eye tracking in software engineering involve code reading, program visualizations and program debugging. Bednarik worked extensively in the area of eye tracking related to program visualizations. Even so, they have only scratched the surface. The two works closely related to this dissertation are the ones by Yusuf et al. [Yusuf, Kagdi, Maletic 2007] and Guéhéneuc et al. [Guéhéneuc 2006; Jeanmart, Guéhéneuc, Sahraoui, Habra 2009]. They were the ones who first conducted studies that used an eye tracker to measure the comprehensibility of class diagrams. The research in this dissertation extends this work by conducting more empirical studies using both traditional and eye-

tracking media to measure the comprehensibility of class diagrams with respect to layouts.

CHAPTER 3

Stereotyped UML Class Diagram Layouts

The research makes a distinction between a UML class model³ and a UML class diagram⁴. A class model consists of all classes and their corresponding relationships and mappings in source code. A class diagram usually consists of a subset of classes from the class model and their relationships styled with layout information. The layout information is not part of the class model. A class diagram need not consist of all possible classes and relationships and need not represent the entire system. An analogy can be made with XML documents and XML style sheets (XSLT) used for pretty printing. Figure 3.1 shows the research focus in shaded grey.

Figure 3.1. A class diagram consists of elements in the design(class) model with layout techniques applied to them. The shaded area shows the research focus.

This chapter describes the layout approaches used for the experiments in this thesis. Three class stereotypes of control, boundary and entity are visually represented

³ UML class model and class model are used interchangeably.

⁴ UML class diagram and class diagram are used interchangeably.

via textual annotations and color. Boundary classes are shown in blue, entity classes are shown in green, and control classes are red. The text annotations are shown above the class name. The following three sections describes each layout in detail.

3.1 Orthogonal Layout

The orthogonal layout is based on general aesthetic criteria [Eichelberger 2003; Eiglsperger, Kaufmann, Siebenhaller 2003; Gutwenger et al. 2003b; Purchase, Allder, Carrington 2002; Purchase, McGill, Colpoys, Carrington 2001] such as minimizing edge crossings, minimizing edge bends, minimizing edge length, maximizing symmetry, and using 90 degree bends. It does not use information about the class stereotype in layout positioning. This is a typical layout (see Figure 3.2) produced by a commercial tool such as Magic Draw [MagicDraw] or Visual Paradigm [VP 2007].

3.2 Multi-cluster Layout

The multi-cluster layout is based on forming multiple clusters in the diagram, where each cluster consists of related classes. The related classes are accountable for specific functionality in the system. Control classes along with their related entity and boundary classes that form a cohesive cluster are grouped closer together. Each cluster has a semantic meaning that is associated to part of a concept or feature in source code and user requirements. Each cluster represents a tightly connected component. This layout depends on the types of relationships that exist between the classes. For example, even though in a generalization hierarchy children are shown immediately below the parent class, in the multi-cluster layout we might position the child closer to another class

it is associated with or dependent on thus highlighting a particular feature in the system. See Figure 3.3. The number of clusters is usually limited to four or five at the maximum, since each cluster consists of four or five classes on average.

3.3 Three-cluster Layout

The three-cluster layout positions classes in a diagram into three clusters for boundary, control, and entity classes. Each stereotyped class category is positioned into a single cluster. See Figure 3.4. The nature of this layout causes it to be wider than the other two layouts in some cases. This cluster separates classes based on their high level architectural importance in the system. It uses the general role of a class in the design and modeling of object-oriented software.

3.4 Summary

It is important to note that it is not always possible to position each type of class in three separate clusters and avoid large amounts of edge crossings. Every attempt was made to maintain aesthetic criteria defined in the literature [Eichelberger 2003; 2005; Purchase, Alder, Carrington 2002]. For example, edge crossings were kept to a minimum and generalizations were drawn to point towards one direction as much as possible.

All the three layouts, orthogonal, three-cluster and multi-cluster layouts display stereotype information via textual annotations (<<control>>, <<boundary>>, and <<entity>>) on top of the class name, as well as color, to control any biases or confounding factors, even though the orthogonal layout does not use the stereotype

information. In order to make a proper comparison, it was necessary for all diagram layouts to exhibit the same design information with only the layout of classes changed thus we were able to isolate layout as a factor to be studied. Note that the clusters on the diagram are not shown in any type of shading or contour, rather they are implicit by the position of the classes.

Figure 3.2, Figure 3.3 and Figure 3.4 show one such example of classes for the orthogonal, multi-cluster and three cluster layouts respectively related to the *PlotterBase* module in the *Hippodraw* system.

Figure 3.2. Orthogonal layout for PlotterBase detailed view

PlotterBase Detailed View [D4]

Figure 3.3. Multi-cluster layout for PlotterBase detailed view

PlotterBase Detailed View [D4]

Figure 3.4. Three-cluster layout for PlotterBase detailed view

CHAPTER 4

The Pilot Study

This chapter presents a pilot study to assess three layout schemes for UML class diagrams. This study was undertaken to determine the feasibility of the approach. The design and running of the user study is described. The results of the study support the hypothesis that layout based on architectural importance is more helpful in class diagram comprehension compared to layouts focusing primarily on aesthetics and/or abstract graph guidelines. The reader is referred to [Juristo, Moreno 2001] and [Wohlin et al. 2000] for details on the experiment design templates and statistical tests used in the experiments conducted in this dissertation. [Oppenheim 1992] is referred to as an excellent source of designing unambiguous questionnaires and interviews for the studies presented here.

4.1 Experiment Overview

This study presents and assesses a layout scheme for UML class diagrams that takes into account the architectural importance of a class in terms of its stereotype (e.g., boundary, control, entity). The assumption is that we are given a UML class diagram that follows the aesthetic criteria mentioned in Chapter 3. Each class in this UML class diagram is annotated with the control, entity, and boundary stereotypes. These are two necessary preprocessing steps to our proposed layout method. Our goal is to reposition classes based on stereotypes that have already been well laid out.

Our hypothesis is that *organizing a class diagram based on architectural importance will help users build better mental models thereby gaining more information about the system*. The next section states the layout approaches used in the study. The design of the user study, results and analyses of the experiment and conclusions drawn are presented next.

4.2 Class Diagram Layouts Studied

In this study, all three layouts namely, orthogonal, multi-cluster and three-cluster, discussed in Chapter 3 are used. Only attributes and their types for classes are shown on the diagrams. Method names of classes are hidden. This was a decision we needed to make since there were many methods. An alternative would be to make visible only those methods that were needed. In future experiments, this will be taken into consideration. In all, we had five class diagrams each presented in one of the three types of layouts styles for a total of fifteen UML class diagrams. Each of the five diagrams represented a particular module of the *Hippodraw* system.

4.3 Study Design

The main purpose of this study is to determine whether *control*, *boundary* and *entity* class stereotypes improve layout and understanding of UML class diagrams while performing comprehension tasks. The study consists of three stages: preprocessing/preparation step, experiment, and analysis of the results (See Figure 4.1).

In the preprocessing step, all participants filled out a questionnaire consisting of eight questions. The reason for this was to collect as much information as possible about

participants, in order to be able to form three equal and fair groups. The reason for choosing three groups is mentioned in the following section. The questions gathered information about programming experience, UML usage and whether or not they were taught (or learnt) a particular UML layout.

Figure 4.1. Setup for the pilot study

There were two exercises targeted at examining the basic knowledge of UML class diagrams. The purpose was to make sure the subjects/participants knew the commonly used relationships and structure of a UML class diagram. Three levels of UML expertise were used: expert, intermediate and basic. Subjects who correctly answered all UML related questions were considered as experts. On the other extreme, subjects who did not get one or both questions right, but had general OO experience and some fundamental understanding of class relationships, were considered to be basic users. Subjects who got an answer partially correct were at the intermediate level. Besides gauging the level of users only on correct answers, we also looked at their experience in industry as well as the approximate number of lines of code they wrote and

projects they worked on. The experiment itself was divided into two parts: comprehension and preference. Finally, the results were analyzed. The following subsections will describe details about the comprehension and preference part of the experiment.

4.3.1 Comprehension/Quantitative Design

This section describes the questions presented in the user study related to comprehension of the system. There were nineteen questions each of which had a time limit attached. The time limits were set based on some preliminary experiments. Each question had a different time limit depending on its complexity. The participants were not allowed to go back to a previous question. We collected both the speed and accuracy of each participant's response.

We subdivided the questions into two sets. In the first set of questions, the participants were asked to match the role of a particular class from a list of choices. Information about the role of the class was presented in the initial ten minute introduction about *Hippodraw*. The second set of questions was related to performing specific maintenance tasks. A scenario was presented along with a list of possible solutions. The participant was supposed to choose one of the listed solutions as their answer. Some dealt with adding a method, whereas others dealt with adding a feature. Both these types of questions were multiple-choice.

4.3.2 Preference/Qualitative Design

In order to gather information about user preferences we presented all three layouts for two class diagrams and asked the participants to choose the one they liked best in terms of comprehension and aesthetics. Another question was related to the use of the colors and text annotations. Each question had a comment box attached. The participants were free to add comments and elaborate on the choice they made and also give any suggestions that they thought would help. We also asked a question regarding the use of curved connectors for dependencies and associations. This part of the study did not have a time limit. However, the amount of time a user spent answering each question was recorded.

4.3.3 Subject System

We conducted the study on *Hippodraw* [Kunz 2006] v. 1.13.1 which was the latest version at the time. *Hippodraw* is an open source data visualization and analysis library. The library can be used for building custom applications in either C++ or Python. It is written in C++ with nearly two-hundred classes and 50 KLOC. Our study focused on the main class hierarchies in the *Hippodraw* system and their most important related classes. However, wherever possible we also showed associations, aggregation, composition, and dependency. The study dealt with fifty classes from *Hippodraw*. To construct the diagrams we reverse engineered *Hippodraw*. The reverse engineering process only extracted generalizations accurately. Next, diagrams for the three types of layouts were manually engineered by inspecting the code for associations and dependencies. All of these diagrams were drawn manually using Microsoft Visio. Since

we had three groups, each group saw the same set of classes but using a different type of UML layout. A subject saw only one version of the diagram causing no learning effect bias in answering questions. The study was designed as a between-subjects experiment.

4.3.4 Subjects

Table 4.1 is a summary of the information about the participants in our study. We gathered twenty subjects to participate in the study. The subjects were a mixture of undergraduate (two) and graduate students (eighteen) in computer science. These twenty participants were classified into one of the three available levels: expert, intermediate and advanced. We then created three groups, each of which had an equal number of experts, intermediates, and basics. For the comprehension (quantitative) part of the study we considered only six of the basic level subjects (to make each group even).

Table 4.1. Summary of participants in pilot study

Number of subjects/participants		20
Number of groups		3
Environment	<i>Controlled</i>	11
	<i>Uncontrolled</i>	9
Subject's Level	<i>Expert</i>	6
	<i>Intermediate</i>	6
	<i>Basic</i>	8
Familiarity with Hippodraw	<i>Very Familiar</i>	0
	<i>Somewhat Familiar</i>	5
	<i>Not Familiar</i>	15

We split the participants into three groups each with equal number of experts, intermediates and basics so that we could have a fair comparison within each group. Each group was given one of the three layouts described above. All subjects were asked

the same questions however the layout of the class diagram shown with respect to each question was different across the three groups.

The study was administered in two forms so that we could accommodate as many participants as possible. Eleven out of the twenty participants were required to meet in an instructional lab at a specific time. We considered this to be the controlled part of the experiment. The other nine participated in the study at their own convenience and not in a lab setting.

Most (75%) of the subjects had no experience with *HippoDraw*. The other 25% were somewhat familiar with *HippoDraw* stating that they did not really use it but had heard about it before. Two of the subjects had reverse engineered *HippoDraw* using Visual Paradigm (as part of a course project). However, they pointed out that they were not very familiar with the design details of the system. All of the participants were familiar with object-oriented design methodology.

4.3.5 Running the Study

The pilot study was setup online so that participants could access it from any location. An open source software system NSurvey [NS] was used to conduct the web based user study. Since we wanted to time each question in the quantitative/comprehension part of the study, a feature was added to the NSurvey system so that each question would have a specified time limit. This also involved making sure that the time is recorded for each question.

Before the subjects were asked any questions, they were given ten minutes to familiarize themselves with *HippoDraw*. A short description was given to them along

with an overview class diagram of the major hierarchies that would be used during the study. Information about the role of certain classes was also presented in the initial ten minute introduction. In this introduction, we also added some information about the UML notations used. This was primarily done to refresh the memory of participants who mainly fell under the basic level of UML expertise. Most of the questions were multiple choice questions. Each question was accompanied by a class diagram.

4.4 Results and Analyses

This section presents the results of the quantitative and qualitative parts of the pilot study. Each section discusses a category of results along with charts for clear understanding.

4.4.1 Distribution of all Answers: Comprehension Questions

The answers considered here are from the comprehension part of the experiment. We did not consider the first question since 50% of the participants failed to answer this question and left it blank. This was due to the fact that they timed out and were not aware of the timeout associated with the question. We noticed this trend only for the first question. All the participants knew that the study was timed in advance but they did not know how much time each question was allotted in advance. The time for each question was shown on the same page that the question was on.

Figure 4.2 and Figure 4.3 show the distribution of answers to eighteen questions at the expert and intermediate levels respectively. The distribution for the basic level is almost the same as that for the intermediate level. The orthogonal layout is also referred

to as the industrial layout and the multi-cluster layout is also referred to as the multiple cluster layout.

Figure 4.2. Distribution of answers at the *expert* level across all 3 layouts

Figure 4.3. Distribution of answers at the *intermediate* level across all 3 layouts

A comparison is made between the number of correct answers across the three layouts for each level: expert, intermediate and basic. The number of correct answers is higher in the group with the multi-cluster layout than in any other layout. Additionally, the number of correct answers is still always more than the number of incorrect answers and the number of blank answers is much less for the group with the multi-cluster layout. An answer is considered blank if it was left unanswered. This could be due to the fact that in the three-cluster layout, the diagrams tended to fit more on the screen compared to the multi-cluster and orthogonal layout. It is hard to reason about blank answers since they could fall into a correct or incorrect category. This trend for the number of blank answers is also seen at the basic level, although the chart is not shown here.

4.4.2 Preference for Stereotypes using Color and Text Annotations

The participants' preferences for color and text annotations are described here. This question was part of the preference section of the experiment. Figure 4.4 shows the distribution of participant preferences.

The goal of this question was to gauge whether colors and text annotations we used were important to point out architectural importance in terms of control, boundary, and entity classes. Experts found that colors or text annotations did not cause any problems or interferences in comprehension. Most of the experts said that the colors helped them in narrowing down the search space to one stereotype.

The participants were also asked whether specific highlighting of the clusters in the multiple cluster layout was useful. All of the participants preferred to have multiple clusters grouped together in light shading.

Figure 4.4. Color and text annotation preference across participant levels

4.4.3 Preference for Comprehension or Aesthetics: Comparing Three Layout Schemes

The question to compare the three layout schemes was part of the preference section of the experiment. Figure 4.5 and Figure 4.6 show this comparison for comprehension and preference respectively. The goal of this question was to find participant preferences for a particular layout considering two aspects: comprehension and preference. Considering only comprehension dealt with pure understanding of the system design. Considering only preference dealt purely with aesthetic criteria such as edge crossings or number of bends. This comparison was based on the exact layouts

given in Figure 3.2, Figure 3.3, and Figure 3.4. We observe that there were more participants who preferred the three-cluster layout (Figure 3.4), however more experts preferred the multi-cluster layout (Figure 3.3). There was only one expert who liked the industrial (baseline) layout for comprehension and preference.

Figure 4.5. Comparison of three class diagram layouts for the purpose of system comprehension

Figure 4.6. Comparison of three class diagram layouts for the purpose of aesthetic preference

4.4.4 Distribution of Maintenance Questions

The questions related to maintenance involved the addition of a method or feature to *HippoDraw*. Figure 4.7 presents the distribution of correct, incorrect, and blank answers for the seven maintenance questions.

Figure 4.7. Results for seven maintenance questions at intermediate and expert levels

We observe that there are no blank answers recorded for the layout with three clusters. The number of correct answers for the layout with three clusters combined with the layout with multiple clusters was higher than the baseline layout. This is an important result and supports our hypothesis that clustering classes with respect to their architectural importance on stereotypes helps a user in understanding a system better than

just an aesthetically pleasing UML class diagram as well as a typical industrial layout class diagram.

4.4.5 Preference for Aesthetics: Highlighted Regions and Curved Connectors

The participants were asked whether specific highlighting of the clusters in the multiple cluster layouts was useful. An example of the diagram shown in the user study is shown in Figure 4.8. All of the participants preferred to have multiple clusters grouped together in light shading. These represented cohesive regions. They found that this helped them in system comprehension. This questions was asked to get an understanding of what users would like to see in a diagram.

A question about the use of curved connectors for dependency and associations was asked in the preference part of the experiment. Two diagrams were presented, one using orthogonal connectors for all relationships and another using curved connectors for associations and dependencies. Figure 4.9 shows the distribution of experts, intermediates and basics regarding this question. The *don't care* response indicates that the diagram with both curved and non-curved connectors looked reasonably good. Although, many participants did not favor the diagram with the curved connectors, their comments indicated that the use of curved connectors for these relationships quickly brought their attention to these parts of the diagram, which is important in certain tasks. This is a preliminary preference result for just one diagram. More preference related studies are needed in this area to make a significant claim of what works well for most users.

Figure 4.8. Multiple clusters highlighted based on architectural importance

Figure 4.9. Number of participants who preferred curved connectors for dependencies and associations in aesthetics

4.5 Summary

This section summarizes all the main results and presents some discussion related to the study. Lessons learned during this experiment are also given. We observe that clustered layouts were more helpful in comprehension tasks. In particular the layout with multiple clusters was most helpful to experts. From this, the conclusion can be drawn that clustered layouts (three-cluster and multi-cluster layouts) were more helpful in answering questions and in understanding the system. Both clustered layouts were useful; however it was difficult to compare both these layouts because they did not fit the same way on the screen. The three cluster layout was wider than the multiple cluster one. The nature of the three cluster layout may have affected the results. The main result is that both clustered layouts were better than the baseline (orthogonal) layout. Our observations show no preference for curved lines but participant comments indicated that curved lines quickly brought attention to the dependency relationship. The curved lines were used as a filtering mechanism to find dependencies quickly.

The experiment was organized so that participants in the controlled and uncontrolled groups were exposed to the same environment conditions. For example, time restrictions were enforced so that participants could not resort to other means of getting information such as searching the web. The environment that the participants took the study also might have affected their performance. Some participants in the controlled lab experiment reported that if they were given the same study to do at their place of preference, (such as their office where they are most comfortable with the mouse and monitors) they may have performed better.

The questions were multiple-choice however they were presented as a drop down list where a participant first had to click to see the list of choices. This caused problems for some users during the study. Having the answers as a list of radio buttons where all are visible at the same time would solve this problem.

Many participants requested that they be made aware of the amount of time left before they timed out on a particular question. This was not implemented as part of the study. This is certainly an important thing to have. If a user was aware of the time in the form of a counter that changed every 10 seconds for example, they would probably choose the best possible answer they thought of at that time.

Another thing pointed out by one expert was about the use of only attributes in the class diagram. The expert stated that it would be much more helpful to have certain methods shown (instead of member variables) with respect to the question asked instead of having attributes, since methods represent the contract between associated classes. Member variables are usually implementation details. This point is considered in other experiments done in this dissertation. A few participants noted that they used the knowledge gained from previous questions to answer questions later on, but this is a subjective claim.

The ultimate goal is to develop guidelines towards a new layout scheme for UML class diagrams with respect to architectural importance. The hypothesis was that it is more important to have an architecturally meaningful UML class diagram rather than an aesthetically pleasing one. This study is the first step towards supporting this hypothesis.

CHAPTER 5

The Effects of Layout on UML Notation and Design Tasks: Replicating an Eye-tracking Study

An empirical study is presented that investigates how stereotype-based layouts impact the comprehension of UML class diagrams. This work continues a previous study using eye-tracking equipment by replicating it using an alternative method. In this study, online questionnaires were used as a means of collecting data. Subjects were given two types of tasks: one addressing UML syntax and the other addressing questions concerning software design. Three different layout strategies are compared. Along with general aesthetics, the layouts are primarily organized based on the class stereotypes of control, boundary, and entity. Besides the answers, a confidence value for each question was collected from the subjects to help validate the categorization of subjects. Results of the study are compared and contrasted to the eye-tracking study done with the same tasks and layouts. Results show a significant improvement in performance in both types of tasks with the multi-cluster stereotyped layouts.

5.1 Overview of the Original Eye-tracking Study by Yusuf et al.

This section describes the original eye-tracking study [Yusuf, Kagdi, Maletic 2007] that is replicated here. This study was conducted by Yusuf et al. on UML class diagrams as an extension to the pilot study discussed in the Chapter 4. The purpose of this study was to assess the comprehension of UML class diagrams using eye tracking.

This study differs from the pilot study in that it uses eye tracking equipment as an implicit means of data collection. Subjects might sometimes forget to report on an answer correctly using questionnaire-based methods. The main factors that are assessed in this study are the effect of layout, stereotypes and color in UML class diagram comprehension tasks. To determine the effects of these factors, they report on what people tend to look at in a class diagram and where, including navigation methods and fixations in UML class diagrams. The Tobii 1750 eye tracker [Tobii 2001] at Kent State University's usability lab was used in their experiment. The eye tracker captures screen coordinates of eye gazes and supports eye movement analysis. A video recording is also made of each subject's study session. The main components of this study are described next.

5.1.1 An Overview of the Study Design

The goal of the study was to understand how humans use information presented in UML class diagrams to perform comprehension tasks. The factors that influenced different UML class diagrams used were layouts, class stereotypes, and color. They used three different layouts namely, orthogonal, three-cluster and multi-cluster (See Chapter 3 for a description of these layout schemes). Color and textual annotations (name of stereotype) were used to visually stereotype classes. The use and description of these three factors were taken from [Andriyevska, Dragan, Simoes, Maletic 2005].

The study consisted of two sets of questions. The first set of questions dealt with UML notational knowledge and the second set of questions dealt with software design knowledge. The purpose of the UML notational knowledge was to understand how a

user performs exploration, explanatory and navigation tasks and to classify subjects as experts or agnostics. The questions related to software design knowledge involved some change requests to the existing design. The purpose of the design questions was to understand how users approach, process and accomplish design tasks. There were 12 UML notational questions and 15 software design questions. The same question was not used for two different layouts to avoid any learning curve bias.

Hippodraw [Kunz 2006] was the subject system being investigated. This was the same one used in the questionnaire-based study [Andriyevska, Dragan, Simoes, Maletic 2005]. Six class diagrams were manually prepared in three different layouts. The study participants were nine computer science subjects and three non-computer science subjects. There was a varying degree of software design and programming experience among these subjects. The three non-computer science subjects had no UML background and no knowledge of software development. Each of the twenty-seven questions was connected to a corresponding UML class diagram based on the three factors of layout, color, and class stereotype information. During the study, subjects answered the question presented to them out loud.

5.1.2 Results and Conclusions from Yusuf et al.

The analysis consisted of accuracy and response time of the answers to the twenty-seven questions. Gaze plots consisting of *saccades*, *fixations* and *scan paths* as well as *heat maps* were used to understand how subjects explore, examine, and navigate through the UML class diagram. Based on the answers of the UML questions, subjects were classified into four categories, UML and design agnostic, UML expert but

inexperienced designer, UML expert and knowledgeable designer and both UML and design experts. The 15 design questions were classified as easy, intermediate, difficult, and challenging based on the way subjects' answered them. A summary of the results is described here.

Subjects explored only parts of the diagram that contained names in the specified question. If a method name was referred to in a question, the subjects tended to look at the parts of the class that show methods/operations. The subjects did not fixate on empty spaces in the diagrams. For questions that involved relationships, fixations occurred at the end of relationship symbols. Subjects who were UML experts and design knowledgeable/experts explored the diagrams from the center out while the UML agnostics and design inexperienced explored the diagrams like they would normally read a document i.e., from top to down and left to right. The use of stereotypes in the form of color and textual annotations were analyzed using gaze plots and video recordings of subjects. They found that UML experts and design knowledgeable/experts used the textual annotations and colors used for stereotypes while answering design related questions whereas they were not used by the design inexperienced.

In order to determine which layout was the most efficient, the average number of fixations of all subjects (except the UML and design agnostic) on each diagram were analyzed. If the total number of fixations is high the diagram is laid out in an inefficient manner. Effort was measured based on these fixations. The results indicate that the three-cluster and multi-cluster layouts outperformed the orthogonal layout in terms of the

amount of effort needed for comprehension. These results support the questionnaire-based study discussed earlier [Andriyevska, Dragan, Simoes, Maletic 2005].

5.2 Focus of the Study

This section describes the focus of our study and research questions asked in this thesis. In a pilot study [Andriyevska, Dragan, Simoes, Maletic 2005], conducted previously, we found class stereotype based layouts to be a promising technique for further evaluation. Additionally, the usefulness of stereotypes for object classification and layout was investigated and assessed using eye-tracking [Yusuf, Kagdi, Maletic 2007]. While neither of these previous studies was comprehensive or statistically conclusive, both seemed to demonstrate that stereotypes are potentially important for comprehension. In the work presented here, we continue this line of research by replicating the eye-tracking study [Yusuf, Kagdi, Maletic 2007] with an alternative methodology. A survey based evaluation study was conducted with a larger number of participants and these results are compared and contrasted to the eye-tracking study.

We conducted an experiment that assesses the comprehension of class diagram layouts for two sets of tasks, namely UML tasks and design tasks. UML tasks address the syntax and structure of visually representing class diagrams whereas design tasks address the design comprehension aspect of the system with respect to maintenance and evolution. The goal of this work is to further validate the results from the eye-tracking experiment [Yusuf, Kagdi, Maletic 2007] and the pilot study [Andriyevska, Dragan, Simoes, Maletic 2005] with a bigger sample and varied experience in the subject system used. Replication with an alternative experimental method will also help better

understand the relationship between different ability levels within the subjects in the two types of tasks examined. The ability level of subjects is determined based on self-assessment questions used in our background analysis and on their performance in both types of tasks. The two main research questions this study tries to address can be stated as follows:

- RQ5.1: Is there an improvement in the comprehension of UML syntax based tasks and general design based tasks for the stereotyped class diagram layouts vs. layouts based on pure aesthetics?
- RQ5.2: How does the subject's ability level affect the comprehension of stereotyped class diagram layouts vs. layouts based on pure aesthetics?

The following sections describe the experimental design, tasks, subject system, subjects and data collection in detail.

5.3 Experimental Design

This section presents details on the logistics of the experiment. The overall design, hypotheses, subject system used, subjects, tasks, data collection and the basic running of the experiment is presented.

5.3.1 Experimental Goal and Hypotheses

The experiment seeks to *analyze* class diagram layouts primarily based on class stereotypes *for the purpose of* evaluating their usefulness in two categories of software comprehension tasks *with respect to* effectiveness (accuracy) and efficiency (time) *from the point of view of* the researcher *in the context of* students at Kent State University.

The high level hypothesis of this experiment is that the layouts based on the class stereotype increase the comprehension of the system. The null hypotheses are formulated below. The alternative hypotheses are easily derived from them.

H_{0u1} : There is no significant difference in UML task comprehension between class diagrams based on the orthogonal layout vs. the three-cluster layout

H_{0u2} : There is no significant difference in UML task comprehension between class diagrams based on the orthogonal layout vs. the multi-cluster layout.

H_{0d1} : There is no significant difference in design task comprehension between class diagrams based on the orthogonal layout vs. the three-cluster layout.

H_{0d2} : There is no significant difference in design task comprehension between class diagrams based on the orthogonal layout vs. the multi-cluster layout.

Table 5.1. Experiment overview for Yusuf et al.'s replication

Goal	Study the comprehension effect of three types of class diagram layouts on two types of software tasks.
Factor / Independent variable	Class diagram layouts with three treatments: orthogonal layout, multi-cluster layout, three-cluster layout
Dependent variables	Quantitative variables: Accuracy, speed, confidence level
Secondary factors	Subject's ability level and task categories

The overview of the experiment is shown in Table 5.1. The main factor being analyzed is the layout of class diagrams. We used three alternatives for the layout as discussed in Chapter 3. The dependent variables are described in Section 5.3.4. While analyzing the results we also looked at secondary factors such as the subject's ability level and the task category (UML and design).

The experiment was conducted as a within-subjects design where all subjects were given all three types of treatments of the factor i.e., class diagram layout. We used this to compare our results to the eye-tracking experiment which also used the same design. Another reason was to gather more data points for each layout.

5.3.2 Subject System

Hippodraw [Kunz 2006] was used as the subject system under investigation. It is an interactive and Python scriptable statistical data analysis application and framework written in C++ with *Qt* for the user interface. *Hippodraw* consists of nearly 96 KLOC in over 600 source and header files. *Hippodraw* does not come packaged with any design documents (class models or class diagrams). However, doxygen documentation is available. We reverse engineered the *Hippodraw* source code using the *srcTools* framework [Sutton, Maletic 2007] to generate the class model. This gave us the classes, associations, generalizations and dependencies between the classes. Some associations were changed to aggregations by manually inspecting the source code. Next, we manually constructed class diagrams using the class model in a UML drawing editor. A total of hundred unique classes are used in this study.

5.3.3 Tasks

The tasks used in this study are identical to the eye-tracking study [Yusuf, Kagdi, Maletic 2007] and we refer the reader there for detailed information about each task and only briefly describe the tasks here. This study consists of two types of tasks: UML tasks and design tasks. The subjects had to provide an answer to a total of 27 questions:

12 UML tasks and 15 design based tasks. The UML tasks were based on four *Hippodraw* modules whereas the design tasks were based on five modules. A total of six *Hippodraw* class modules were constructed based on related functionality which resulted in 18 diagrams (6 diagrams * 3 layouts). Three of the modules were common between UML tasks and design tasks. In this study, the class diagram layouts used are the same as described in Chapter 3. See Figure 5.1 for an example of the three layouts used.

Three of the modules consisted of one class diagram drawn in three different layouts (9 diagrams). The other three modules had 2 diagrams drawn in three different layouts (18 class diagrams). The diagrams were manually engineered in a UML drawing editor using general aesthetic criteria [Ambler 2002; Eichelberger 2003; Eiglsperger, Kaufmann, Siebenhaller 2003; Gutwenger et al. 2003b; Purchase, Allder, Carrington 2002; Purchase, McGill, Colpoys, Carrington 2001] and stereotype information (in the case of the three-cluster and multi-cluster layout). The number of classes used for each of the modules range from a minimum of 12 (*XmlNode* module) to a maximum of 21 (*PlotterBase* module) classes.

Figure 5.1. Class diagrams for the *Python Wrappers* module shown in three different layouts: orthogonal, three-cluster and multi-cluster

An example of a UML task would be asking the subject to select all the classes involved in dependency or to identify the kind of relationship between two classes. These types of tasks depend on the UML syntax/notation used for class diagrams.

A design related task required the subject to analyze the class diagram to answer specific questions about understanding *Hippodraw*. One example of a design task used was: Which class controls the active window of an application? The subjects did not need to be an expert in *Hippodraw*'s design and/or implementation to answer these questions. The answers to questions could be found by analyzing the classes, relationships, attributes, methods, and stereotypes.

Since this is a within-subjects study, the same task is not asked for more than one layout. This eliminates any learning bias involved in answering the same question twice. Instead, similar questions were asked for the three layouts. Consider the following three questions from the *PythonWrappers* module. Each of these questions are similar in nature that allows analysis of subject's performance across three layouts. The class diagrams that accompany these questions contain the same information with the exception of the layout.

Table 5.2. Three design tasks asked for the PythonWrappers module. The layouts used for these questions are shown in Figure 5.1

ID	Layout	Question Text
Q13	Orthogonal	Select the class that a python wrapper uses to access data in the class NTuple
Q18	Three-cluster	Select the class that is a python wrapper for a class with the method name adduct.
Q23	Multi-cluster	Name the entity class that is responsible for storing data

5.3.4 Data Collection

We used three online questionnaires to gather data in this experiment. The first questionnaire collected background information about the subjects. This information is presented in Section 5.3.5.

Table 5.3. Debriefing Questionnaire

ID	Question	Mapping
1	I had sufficient time to complete the questions	1-5
2	I think the questions were difficult to answer	1-5
3	The questions were clear to me	1-5
4	I was able to understand information in the class diagrams	1-5
5	I think the questions were realistic	1-5
6	I found UML class stereotypes useful in answering questions	1-5
7	Did you concentrate on the spatial layout while answering questions?	Yes = 1 No = 0 Not sure = 2

The second questionnaire consisted of the actual study tasks. Each task (UML and design) was given a score. We calculated the accuracy of answering UML tasks and design tasks based on the score. The speed i.e., time taken to complete each task was also recorded. Besides the accuracy and speed, we also collected a confidence level of the subject's answer for each task. The confidence level was on a Likert scale from 1 (not confident) to 5 (very confident). Finally, the third questionnaire was a debriefing questionnaire that collected data about the task and stereotypes used. See Table 5.3 for questions.

5.3.5 Subjects

We gathered twenty-nine subjects (14 undergraduate students and 15 graduate students) to participate in this experiment. The undergraduate students were within the

age range of 18-25 years. Ten of the graduate students were between 25-35 years and five were between 35-45 years. They were all from the computer science department at Kent State University. There were 23 males and 6 females. Five of the subjects had used class diagrams in both academia and industry while the rest had theoretical knowledge of UML and applied it in an academic setting, typically in software engineering courses.

The subjects were informed that the purpose of the study was to understand how people interpret class diagrams (not their UML expertise). They were not aware of the experiment's hypotheses or of the different layouts used. They were also instructed to answer the questions from the point of view of a maintainer trying to understand the system. Figure 5.2 shows the descriptive statistics of our sample population gathered from the background questionnaire. We collected information about design and programming skills, number of years of experience in general programming and in OO programming and familiarity of *Hippodraw*. The programming and design skills were on a scale from 1 to 5, the others were on a 1 to 4 scale. A low rating indicates low experience, skill and familiarity.

Figure 5.2. Descriptive statistics of subjects' background for the Yusuf et al replication study

We can see from the figure that there was a large difference in the familiarity of *Hippodraw* among the participants. Most of the subjects (with the exception of 1) were not familiar with the design of *Hippodraw*. The subjects reported their self assessment of programming and design skills. This is correlated with the accuracy of UML and design questions in Section 5.4.

5.3.6 Study Instrumentation

The study was conducted online. A fixed amount of time up to one minute was allotted to each question. We did this to keep the subjects on task and to replicate the timing aspect of the eye-tracking experiment as close as possible. A couple of days before the experiment, subjects were asked to go through a class diagram tutorial. A short description of class stereotypes and their graphically representation was given. They were also informed of the colors used to differentiate between different class stereotypes. The tutorial was optional, however, all subjects with an exception of a few participated in the tutorial. The purpose of the tutorial was to make sure all subjects were on the same page with respect to understanding information presented in the class diagrams used in the study.

After the tutorial, the study was completed in one sitting. Subjects were first presented with a set of instructions and a one screen description (on a 17" monitor) of the *Hippodraw* system. We recommended that the subjects take the study on a 17" monitor in order to view most of the diagram on one screen with minimal scrolling. All the questions were multiple-choice. They were made aware that each question had a time limit attached to it and that their responses were timed. The time limit for each question

was shown on the top of the page near the question. If time ran out before the subject could answer the question, the question was considered to be left blank or the answer chosen at that point of time was recorded. The instructions also stated that they will have 10 seconds to rate each answer they provide. The subjects were not allowed to go back and change an answer once they move to the next question. However, if a subject felt that they answered incorrectly they have the option of choosing a low confidence value for that answer.

The following information was presented for each task: a question, answer choices and a class diagram in one of the three possible layouts. The subjects were asked to choose the answer for the question with respect to the class diagram. After all the tasks were completed, a debriefing questionnaire was presented for the subjects to complete. This concluded the experiment from the subject's viewpoint.

5.4 Experimental Results and Analyses

This section presents the results of this experiment. We first discuss the parts common to this study and the eye-tracking study: classification of subjects and questions. Next, we discuss exclusive observations pertaining to this study: effect of layout on UML and design tasks as well as the effect of ability and layout together. Confidence levels are correlated with performance and skill level of subjects. Finally, we present results from the debriefing questionnaire.

5.4.1 Subject Classification

The accuracy and speed of 12 UML tasks and 15 design tasks were analyzed. Figure 5.3 and Figure 5.4 show the accuracy and speed for all 29 subjects. None of the subjects answered all UML questions and all design questions correctly.

Figure 5.3. UML and Design task scores across all twenty nine subjects. The maximum possible score is 42 for UML and 18 for design. The subjects are first sorted by UML score and then by design score.

Figure 5.4. Time (in minutes) taken to complete UML and Design tasks across all twenty nine subjects. The order of subjects is the same as Figure 5.3 to have a side by side comparison.

Based on the performance of subjects in answering the questions, we classified them into eight groups. This was also done in the eye-tracking study [Yusuf, Kagdi, Maletic 2007]. Table 5.4 shows the categories. The main groups are: Agnostic (A), Inexperienced (I), Knowledgeable (K) and Expert (E). UML scores in the ranges of [0,23],[24,29] and [30,42] were mapped to the agnostic, knowledgeable and expert category respectively. Design scores in the ranges of [0,3],[4,6], [7,10] and [11,18] were mapped to the agnostic, inexperienced, knowledgeable, and expert category respectively. The eight categories of interest to us are shown in the table.

Table 5.4. Categorization of subjects for UML and Design tasks. UML and design tasks are shown by an U and D respectively.

Design Categories						
UML Categories		DA	DI	DK	DE	Total
	UA	6	3	2	x	11
	UK	x	x	2	5	7
	UE	x	2	5	4	11
	Total	6	5	9	9	29

- The **UADA** group represents subjects with little knowledge of UML and software design. There are four subjects (K, L, M, O, Y, S) in this group. They took between 1 and 8 minutes to complete the study.
- The **UADI** group represents subjects with little knowledge of UML and some basic design knowledge. There are three subjects (V, Q, AA) in this group. They took between 11 and 15 minutes to complete the study.
- The **UADK** group represents subjects with little knowledge of UML and more knowledgeable in design. There are two subjects (C, A) in this group. They took 9 and 13 minutes to complete the study.
- The **UKDK** group represents subjects knowledgeable in UML and design. There are two subjects (R, T) in this group. They took 14 and 16 minutes to complete the study.
- The **UKDE** group were knowledgeable in UML and experts in software design. There are five subjects (F, G, Z, H, J) in this group. They took between 10 and 17 minutes to complete the study.
- The **UEDI** group were very proficient in UML with basic design knowledge. There are two subjects (I, U) in this group. They took 13 and 20 minutes to complete the study.

- The **UEDK** group were very proficient in UML and knowledgeable in software design. There are five subjects (B, N, D, P, E) in this group taking between 12 and 21 minutes to complete the study.
- The **UEDE** group were experts in both UML and design. Four subjects (W, AC, AB, X) fall into this category. They took between 9 and 19 minutes to complete the study.

The UML knowledgeable (UK) category was not present in the eye-tracking experiment due to the small focus group of subjects in that study. The classification of subjects into these groups shows a varying expertise in UML and software design skills. We ran a correlation test to determine if there is a match between self assessed design skills (See Figure 5.2) and UML and design scores. The *Spearman* rank correlation between self assessed design skills and design task scores ($r_s = 0.39$ $p\text{-value}=0.01$) indicates a significant positive correlation between the two. There is also a positive correlation between programming skills and design task scores ($r_s = 0.34$ $p\text{-value}=0.03$) and programming skills and UML task scores ($r_s = 0.32$ $p\text{-value}=0.04$). No correlation was found between UML Scores and Design skills ($r_s = 0.212$ $p\text{-value}=0.135$). This is not surprising since there are subjects who are experts in design but don't use UML on a regular basis.

5.4.2 Question Classification

We classified the UML and Design questions (tasks) based on the number of correct answers in each. Figure 5.5 shows the dispersion and skew of the number of correct answers in UML and design questions.

Figure 5.5. Box plot of the answer distribution between UML and design tasks

UML questions that were answered correctly in the ranges of [0%,59%), [59%,69%), [69%,76%), [76%,100%) were classified as challenging, difficult, intermediate and easy respectively. All UML questions were classified as easy in the eye-tracking study. Design questions that were answered correctly in the ranges of [0%,31%), [31%,43%), [43%,55%), [55%,100%) were classified as challenging, difficult, intermediate and easy respectively. We used interquartile analysis to derive these ranges. Table 5.5 shows the question classification in this study. The design questions in boldface font are classified at the same level in both this and the eye-tracking study. With respect to design questions, there is a 100% match in the challenging category, a 50% match in the easy and difficult category and a one question match in the intermediate category. From the mismatched items, four of them were placed into a higher category in this study and three were placed into a lower category.

Table 5.5. UML and Design Question Classification

Level	UML Questions	Design Questions
Easy	8, 9, 12	14, 15, 16 , 26
Intermediate	2, 4, 5, 6, 11	17, 19, 21, 22
Difficult	1, 3, 10	13, 24, 25, 27
Challenging	7	18, 20, 23

We did not expect UML questions to fall into the challenging category but we did have one such case. Figure 5.6 shows the task with the diagram. There were only two subjects (F, P) who answered this question correctly. The subject was asked to select the classes derived from Observer. The correct answer is ViewBase, PlotterBase and DataRep. But this is not very clear from the diagram since DataRep is positioned at the same level as its parent. Another reason for this question to get challenging (even though we feel it was supposed to be easy) was the fact that DataRep derives from both Observer and Observable. If DataRep was positioned at a lower level, it might have generated more correct answers. The multiple inheritance might have also played a role in answering this question. We cannot be sure as to why this happened. This is one example where the three-cluster layout did not perform well. This case is revisited in Section 5.4.3.

Figure 5.6. The class diagram in three cluster layout used for question 7: Which classes are derived from Observer? This is best viewed in color.

We use the question classification to generate a weighted score for UML and design tasks. Easy questions were given a lower weight and difficult questions were given a higher weight. The total weight equals 1. In the eye-tracking study, the classified questions were used to compare the effort needed based on the average number of fixations. We are not able to do this analysis due to the lack of eye tracking data. Instead we determine the layout performance based on accuracy shown in the next two sections. The next two sections focus on answering RQ5.1 and its corresponding hypotheses.

5.4.3 Effect of Layout on UML Tasks

This section analyzes the accuracy of UML tasks for all subjects and determines if the layout had any effect on task accuracy. The first and second hypotheses (H_{0u1} and H_{0u2}) seek to determine the effects of the orthogonal layout vs. the three-cluster layout and the orthogonal layout vs. the multi-cluster layout respectively for UML tasks. We use the paired Wilcoxon non-parametric test to determine the better layout. We conduct a pair wise comparison between the three layouts. The results for UML tasks are shown in Table 5.6 with the box plots shown in Figure 5.7.

Table 5.6. Results of the pair wise Wilcoxon test for UML tasks. Significance is shown in bold. $\alpha = 0.05$. For the 1-tailed values, the direction is given by the order of treatment pairs i.e., first < second.

Treatment Pairs	Z-statistic	<i>p-value</i> (2 tailed)	<i>p-value</i> (1 tailed)	Diff Median
Orthogonal and three-cluster	1.43	0.1533	0.923	1.0
Orthogonal and multi-cluster	-2.90	0.0037 *	0.0019 *	-3.0
Three-cluster and multi-cluster	-3.98	<0.0001 *	<0.0001 *	-3.5

Results indicate a significant difference between the orthogonal layout vs. the multi-cluster with the multi-cluster layout performing better (1-tailed *p-value*= 0.0019). No significant difference was found between the orthogonal layout and the three-cluster layout. One of the reasons could be due to Question 7 (based on the three cluster layout) which was classified as challenging since it was not answered correctly by most participants. To determine this, we excluded Question 7 and its corresponding questions from the orthogonal and three-cluster layout from the analysis and ran the paired Wilcoxon test again on all the data. This gives a significant difference (1-tailed *p-value* =

0.0004) between the orthogonal and three-cluster layout with the three cluster layout outperforming the orthogonal layout.

Figure 5.7. Box plot of UML scores across layouts for the Yusuf et al. replication study

There is also a significant difference between the three-cluster layout and the multi-cluster layout with the multi-cluster layout outperforming the three-cluster layout (1-tailed p -value < 0.0001). We did not formulate any hypotheses between the three-cluster and multi-cluster layouts so this is a new observation.

5.4.4 Effect of Layout on Design Tasks

This section analyzes the accuracy of design tasks for all subjects and determines if the layout had any effect on task accuracy. The third and fourth hypothesis (H_{0d1} and H_{0d2}) seek to determine the effects of the orthogonal layout vs. the three-cluster layout and the orthogonal layout vs. the multi-cluster layout respectively for design tasks.

Similar to UML task analysis, we use the paired Wilcoxon non-parametric test to determine the better layout. We conduct a pair wise comparison between the three layouts. The results are shown in Table 5.7. Figure 5.8 shows the box plots of the design tasks in the three layouts.

Results indicate a significant difference between the orthogonal layout vs. the three-cluster layout (1-tailed p -value = 0.005) with the three-cluster layout performing better. A significant difference between the orthogonal layout and the multi-cluster layout was also found (1-tailed p -value=0.0031) with the multi-cluster layout performing significantly better than the orthogonal layout. No significant difference was found between the three-cluster layout and the multi-cluster layout. This suggest that for the design tasks, both three-cluster and multi-cluster layouts performed equally well.

Table 5.7. Results of the pair wise Wilcoxon test for design tasks. Significance is shown in bold. alpha = 0.05. For the 1-tailed values, the direction is given by the order of treatment pairs i.e., first < second.

Treatment Pairs	Z-statistic	<i>p</i> -value (2 tailed)	<i>p</i> -value (1 tailed)	Diff Median
Orthogonal and three-cluster	-2.58	0.0099 *	0.005 *	-0.20
Orthogonal and multi-cluster	-2.74	0.0062 *	0.0031 *	-0.20
3-cluster and multi-cluster	0.11	0.9138	0.5431	0.00

Figure 5.8 . Descriptive statistics of design scores across layouts for the Yusuf et al. replication study

5.4.5 Effect of Ability and Layout on UML and Design Tasks

This section addressed our second research question (RQ5.2). It investigates if the subject's ability level affects the comprehension of stereotyped or orthogonal class diagram layouts. The ability of a subject is the UML group (UA, UK, UE) and design group (DA, DI, DK, DE) they belong to. Figure 5.9 and Figure 5.10 show the average UML and design scores with respect to the three layouts used. We analyze the UML tasks and design tasks separately.

The UML task results, indicate that in all the three UML categories, the multi-cluster layout performed better than the orthogonal and the three-cluster layouts. The three-cluster layout was the second best layout in all three UML categories with the exception of the UA category.

For design tasks, we based our comparison on low and high design abilities. The low design ability subjects were from groups DA and DI. The groups DK and DE were combined to form the high design ability group. From Figure 5.10, we see the performance for low design ability subjects to be higher for multi-cluster layouts. The same trend is seen in the high design ability group. There is not much difference between the multi-cluster layout and three-cluster layout for the high design ability group. This relates to the discussion in the above section where it was statistically shown that three-cluster and multi-cluster layouts were equally effective.

Figure 5.9. UML task scores across layouts for the Yusuf et al. replication study

Figure 5.10. Design task scores across layouts combined into low (DA+DI) and high (DK+DE) design ability for the Yusuf et al. replication study

We did not conduct a 2-way *ANOVA* to determine interaction between ability and layout due to low sample size ($n=29$) and non-normality of the UML task scores. This will be conducted in future experiments with a larger sample.

5.4.6 Confidence for Layouts in Subject Categories

We collected the confidence level for each question to analyze the way subjects' rate each answer and whether this correlates with the subjects' self assessment of their design and programming skills. The *Spearman* rank correlation between design skills and the average design task confidence level for all UML questions ($r_s = 0.46$ $p\text{-value} = 0.009$) indicates a significant positive correlation between the two. However, no correlation was found between design skills and average UML task confidence for all design questions.

The *Spearman* rank correlation between UML Scores and the average UML confidence level for all UML questions ($r_s = 0.76$, $p\text{-value} < 0.001$) indicates a significant

positive correlation between the two. The same is shown between design scores and average design task confidence level for all design questions ($r_s = 0.79$, $p\text{-value} < 0.001$). Figure 5.11 shows the confidence level for subject categories. High levels of ability in UML and Design result in higher confidence ratings.

Figure 5.11. The average confidence level for UML and Design tasks at each ability level for the Yusuf et al. replication study

5.4.7 Debriefing Questionnaire Results

We present the results of our debriefing questionnaire here. Figure 5.12 shows the rating of the questions 1 through 6 on a Likert scale (1= strongly agree, 5=strongly disagree). Half of the subjects stated that the time given was not sufficient. These were subjects that fell into the low ability groups. We needed to time the questions to make a fair comparison to the eye-tracking study which did not involve any interruptions or distractions. The data also shows that the questions were considered somewhat difficult to answer. Again this rating was more prevalent in the low ability groups.

Overall, the subjects agreed that the questions were clear, realistic and the information in the class diagrams was understandable. They also considered stereotypes to be helpful while answering questions.

Figure 5.12. Box plot of debriefing questionnaire results for the Yusuf et al. replication study

Question 7 from the debriefing questionnaire asked the subjects if they concentrated on the spatial layout while answering the questions. Only seven subjects said that they did. However, this number is very subjective since even though they might have concentrated on the spatial layout, they might not have been aware of it. An eye-tracking analysis would have been useful here.

5.5 Discussion

The results of this experiment show a significant improvement in subjects' performance with the multi-cluster layout. In particular, for UML tasks, the multi-cluster layout was the best layout when compared to the orthogonal layout and three-cluster

layout. We also compared the three-cluster layout with the multi-cluster layout and found the multi-cluster layout to be significantly better in performance for UML tasks.

For design tasks, the three-cluster layout and the multi-cluster layout outperformed the orthogonal layout. There was no significant difference in performance between the three-cluster and multi-cluster layout for design tasks.

In addition to the above results, we find that the multi-cluster layouts resulted in better performance at each subject ability level (agnostic, inexperienced, knowledgeable, expert). The three-cluster layout was the second best layout with the orthogonal considered to have the worst performance. For design tasks, in particular, there was no difference in performance between the three-cluster layouts and multi-cluster layouts in each subject group. This is consistent with the analysis without considering ability.

The subjects' confidence level for the tasks correlate positively with their design skills. We observed a positive correlation between task scores and design and programming skills. Since the subject's categorization into ability level groups was based on the task scores, we wanted to validate this against the subjects self-assessment of programming and design skills.

Subjects in higher ability groups (UK, UE, DK, DE) were more confident of their answers compared to lower ability groups (DA, DI, UA) in both UML and design tasks. Finally, our qualitative assessment of the debriefing questionnaire revealed that the subjects found class stereotypes helpful in answering the questions. We did not find any significant effect between time taken to complete the task and the layouts used.

5.6 Similarities and Differences with the Eye-tracking Study by Yusuf et al.

We now compare and contrast our study with the eye-tracking study. The common goal between these two studies was to find the layout that is most effective for comprehension tasks. One major difference is the method of data collection. We did not use eye tracking equipment, instead online questionnaires were used. The prediction, based on our previous work, was that the clustered layouts would result in better performance. This is proved using statistical significance tests in this study. The eye-tracking study did not produce such significance. Our subject classification resulted in the UML knowledgeable (UK) category that didn't exist in the eye-tracking experiment.

Another difference between our study and the eye-tracking study is the subjects' familiarity with *Hippodraw* and the sample size. In this study, we used a bigger sample (n=29). This is a larger sample compared to our pilot study [Yusuf, Kagdi, Maletic 2007] (20 subjects) which was run as a between-subjects study. The eye tracking study had 9 subjects. Several subjects in the eye-tracking study were familiar with *Hippodraw*'s design or had used it before. In our study we have a more varied sample with most subjects not familiar with *Hippodraw*. Another difference is that the eye-tracking study uses information about the average number of fixations for each question to determine the effort required by subjects. It then compares this effort with the difficulty level of each question to determine if the effort is at the same, higher or lower level. They find the most effort was required with the orthogonal layouts. This differs from our study, where we determine the usefulness of stereotyped layouts using the accuracy of answers and statistical significance tests.

5.7 Threats to Validity

Internal validity refers to the presence of other factors besides the main factor that might have an effect on the results. Since this was a within-subjects experiment we had to make sure that there was no learning effect involved when comparing the results of three layouts. We address this by asking a very similar question for each of the three layouts to have a fair and unbiased comparison between them. The questions were presented to each subject in a randomized order to further reduce any learning effect that might occur. Since the experiment was part of the subjects' grade in a course, they were sufficiently motivated to do well.

External validity deals with generalizing our results. We used students as subjects in our study. Many of the subjects had worked with UML in academia and industry. The high ability group of students had real world experience in designing and maintaining software systems. We can liken this group of subjects to mid-level to senior level developers. The subject system we used is a real life system not a toy application. Subjects tend to agree that tasks were realistic and typical of ones they would ask themselves during maintenance.

To ensure conclusion validity, we use the non-parametric paired Wilcoxon statistical test to determine significance of stereotyped layouts vs. the orthogonal layout due to the small sample size.

5.8 Summary

This study measures the effect of stereotyped class diagram layouts on two types of comprehension tasks. The first set of tasks dealt with UML syntax and the second set

of tasks dealt with elements of design. Results show a significant improvement in performance accuracy when multi-cluster layouts were used, for both UML tasks and design tasks. The second best layout was the three-cluster layout for UML tasks with the orthogonal layout having the worst score in both UML and design tasks. The three-cluster and multi-cluster layouts performed equally well for the design tasks. We do not claim to generalize our results to different types of tasks. This would require further empirical analysis.

These results repeat and add to the findings of the eye-tracking study (Section 5.1) and the pilot study in Chapter 4, which suggests that stereotyped layouts have a positive effect on the comprehension of class diagrams. This experiment further validates the results of an eye-tracking experiment on the same set of class diagram layouts and tasks. This shows that eye-tracking and online questionnaires are complementary techniques of obtaining comprehension performance.

CHAPTER 6

The Effects of Layout in Software Comprehension Task Categories

The results of a controlled experiment assessing the effects of different layout strategies on the comprehension of UML class diagrams of two software systems is presented in this chapter. Six different categories of software comprehension tasks, with varying degrees of difficulty, are used to assess the layouts. Each task consists of several questions aimed at measuring the comprehensibility of a layout. The study involved 45 participants of varied experience in software design and programming ability. A report on the quantitative analysis of accuracy, speed, confidence level and preference of solving the tasks is given. Results indicate that clustered layouts demonstrate significant improvement in subject accuracy and speed in solving the problems in a majority of tasks.

6.1 Overview and Research Questions

This study continues our previous work [Andriyevska, Dragan, Simoes, Maletic 2005; Sharif, Maletic 2009b] of assessing the usefulness of class stereotypes (control, boundary and entity) in class diagram layout. Along with general aesthetics [Eichelberger 2003; Eiglsperger, Kaufmann, Siebenhaller 2003; Gutwenger et al. 2003b; Purchase, Alder, Carrington 2002; Purchase, McGill, Colpoys, Carrington 2001], the layouts are primarily organized by class stereotypes of control, boundary, and entity. Our initial pilot study [Andriyevska, Dragan, Simoes, Maletic 2005] compared three class

diagram layouts (orthogonal, multi-cluster, 3-cluster) and showed that two of the clustered layouts (multi-cluster and three-cluster) were more helpful in class diagram comprehension. Yusuf et al. continued this work by conducting an eye-tracking study [Yusuf, Kagdi, Maletic 2007], to understand how people explore information in stereotyped class diagrams using the stereotyped layouts with different tasks. The use of layout, color, and class stereotypes were all assessed to determine their effectiveness in program comprehension. In Chapter 5, we replicated the eye-tracking study via online questionnaires and got comparable results. The results of the replicated study were published in [Sharif, Maletic 2009b] that statistically showed that the multi-cluster layout does indeed outperform the orthogonal and three-cluster layouts in two types of tasks: UML syntax related tasks and design related tasks.

In this chapter, we report the results of a third experiment in our investigation of the effectiveness of stereotyped class diagram layouts. This study includes a larger number of participants, uses two systems to validate our hypotheses and also includes a larger number of class diagrams. In addition, the main goal of this study is to investigate the effect clustered stereotyped layouts have on six types of task categories ranging from easy to challenging. In other words, a finer-grained analysis is conducted. Our previous studies did not focus on task categories. They mainly dealt with very basic overview tasks such as the identifying the role of a class in a diagram. The two main research questions this paper tries to address can be stated as follows:

- RQ6.1: Which software comprehension task categories benefit most from stereotyped class diagram layouts?

- RQ6.2: Do stereotyped layouts in each task category help design experts and design novices in the same way?

The stereotyped layouts used are described next. The experimental design is discussed in Section 6.3. Section 6.4 analyzes the results and reports the findings of the experiment. Section 6.5 addresses the threats to validity followed by the conclusions.

6.2 Stereotyped Layouts

Three class stereotypes of control, boundary and entity are visually represented via textual annotations and color. This study uses two types of layouts: orthogonal and multi-cluster. These layouts are described in Chapter 3. The selection is based on our previous work [Sharif, Maletic 2009b], that suggests the multi-cluster layout outperforms the three-cluster layout. The three-cluster layout is not used here. .

6.3 Experimental Design

This section presents details on the logistics of the experiment. The overall design, hypotheses, subject systems used, subjects, tasks, data collection and the basic running of the experiment is presented.

6.3.1 Experimental Goal and Hypotheses

The experiment seeks to *analyze* two class diagram layouts primarily based on class stereotypes *for the purpose of* evaluating their usefulness in six categories of software comprehension tasks *with respect to* effectiveness (accuracy) and efficiency (speed) *from the point of view of* the researcher *in the context of* students at two

universities. The detailed null hypotheses are given below. The alternative hypotheses are 1-tailed predicting the multi-cluster layout performs better.

H_{0a} : There is no significant difference in comprehension *accuracy* between orthogonal layouts and multi-cluster layouts for six categories of tasks. $\mu(\text{Accuracy}_{\text{ortho}}) = \mu(\text{Accuracy}_{\text{multi}})$

H_{0s} : There is no significant difference in comprehension *speed* between orthogonal layouts and multi-cluster layouts for six categories of tasks. $\mu(\text{Speed}_{\text{ortho}}) = \mu(\text{Speed}_{\text{multi}})$

H_{0e} : Design experience does not significantly interact with the layout type (orthogonal or multi-cluster) to have an effect on task comprehension or speed.

The overview of the experiment is shown in Table 6.1. The main factor being analyzed is the layout of class diagrams with two treatments: orthogonal and multi-cluster. While analyzing the results we also looked at secondary factors such as the interaction effects subjects' design experience (design experts vs. design novices) has on accuracy and speed.

Table 6.1. Experiment overview for task categories study

Goal	Study the comprehension effect of two types of class diagram layouts in six software task categories.
Main Factor	Class diagram layouts with two treatments: orthogonal layout, multi-cluster layout
Dependent variables	Accuracy, speed, confidence level
Secondary factors	Design experience (experts and novices) Subject systems (Qt and wxWidgets)

Table 6.2. A 2*2 factorial design with two systems and two layouts for the task categories study

	Group A (N=27) N_{experts}= 14 N_{novices}= 13	Group B (N=18) N_{experts}= 9 N_{novices}= 9
First Run	Qt with multi-cluster layout on Day 1	Qt with orthogonal layout on Day 1
Second Run	wxWidgets with orthogonal layout on Day 2	wxWidgets with multi-cluster layout on Day 2

The experiment is split into two parts: comprehension and preference. The comprehension part consists of twenty-two questions in six different task categories (See Section 6.3.3) for task category description. The preference part consists of rating two class diagram layouts for aesthetics and comprehensibility with respect to two questions from the comprehension part of the experiment. That is, the preference rating is tied to a question from a task category. We believe this gives a better rating versus asking a user to rate a diagram with respect to no task.

The comprehension part of the experiment was conducted as a within-subjects 2*2 factorial design where all subjects in each group were tested on both types of layouts on two different systems. See

Table 6.2 for experiment setup. The layout type and system were switched for Group B. The same question was not asked for the same layout in the same system to avoid learning effects; instead a similar question was asked in a different system. The preference part of the experiment was the same for each group i.e., both groups rated the orthogonal and multi-cluster layout for each system with respect to two questions. The experiment was conducted in two runs, each on different days. The two runs were done on different (not necessarily consecutive) days to reduce sequencing effects.

6.3.2 Subject Systems

Two comparable GUI toolkits were used to test our hypotheses: *Qt*⁵ and *wxWidgets*⁶. The number of unique classes used in *Qt* and *wxWidgets* are 122 and 109 respectively. Both *Qt* and *wxWidgets* have a good set of documentation available online along with a basic hierarchy of their class model. However, neither system includes class diagrams as documentation with the exception of *doxygen* documentation. The *srctools* framework [Sutton, Maletic 2007] was used to generate the class model. The diagrams in two different layouts were then manually engineered in a UML drawing editor by inspecting the code and online documentation for entity, boundary and control classes that work together towards a specific feature or functional requirement.

6.3.3 Task Categories

This study consists of six types of tasks: *Reading* (4), *Overview* (9), *Impact Analysis* (3), *Bug Fix* (1), *Feature Addition* (3), and *Refactoring* (2). See Table 6.3 and Table 6.4. There were twenty-two questions spread across these six task categories.

- *Reading* task questions dealt with finding paths between classes, reading the class stereotype information from the diagram, or identifying relationships and classes involved. This was the least difficult of all the tasks. No critical thinking was required to perform these tasks.

⁵ <http://www.qtsoftware.com>, v. 4.3.3, ~ 729 KLOC

⁶ <http://www.wxwidgets.org>, v. 2.8.7, ~ 609 KLOC

- *Overview* task questions consisted of 9 questions that involved understanding the system at a high level such as identifying the role of a class/method or identifying high level functionality in classes.
- *Impact Analysis* task questions asked to identify classes that need to be changed if some functional requirement was added or modified. These tasks impacted both classes and methods. Even though impact analysis tasks are well supported by static analysis tools, we include this type of task to determine the effect layout might have in determining classes impacted by a design change at the UML class diagram level.
- The *Bug Fix* question involved presenting the subject with a real bug description taken from the bug tracking repository (bug id 85876 for *Qt* and 1168331 for *wxWidgets*). The objective was to identify classes that needed to be looked at to fix the bug.
- *Feature Addition* task questions involved presenting a new feature addition scenario. This mainly involved sub-classing operations in order to add the new functionality.
- *Refactoring* task questions asked to improve the design of the existing class hierarchy. Subjects were scored based on whether they chose the correct methods/classes for the pull up method/field or collapse hierarchy refactorings. They were not specifically asked for the names of the refactorings.

A total of eight class modules were constructed based on related functionality which resulted in thirty-two diagrams (eight diagrams * two layouts * two systems). Aesthetic criteria [Eichelberger 2003] are adhered to whenever possible. However, sometimes we prioritized on cluster formation in multi-cluster layouts over aesthetics. The multi-cluster layouts in each module (except the Widgets module which contained five clusters) contained four clusters for both *Qt* and *wxWidgets*.

Each of the task categories were rated by an expert in terms of their difficulty level. The *Reading* and *Refactoring* tasks were considered to be easy. *Overview* tasks were moderately difficult. *Impact analysis* tasks were considered to be more difficult. The *Bug Fix* and *Feature Addition* tasks were classified as challenging. The classification was based on the very nature of the task itself and the questions involved. The *Bug Fix* and *Feature Addition* tasks needed the subject to understand and process a lot more of the diagram to answer the question correctly. Answers to all questions could be found by analyzing the classes, relationships, attributes, methods, and stereotypes in the diagram.

Table 6.3. Modules per task category used in *Qt* and *wxWidgets* (wx). Each module is represented by a class diagram in two layouts: orthogonal (ortho) and multi-cluster (multi). The difficulty level is shown near each task category.

Modules	System	Software Task Categories						Preference Questions
		Reading (Easy)	Overview (Moderate)	Impact Analysis (Difficult)	Bug Fix (Challenging)	Feature Addition (Challenging)	Refactoring (Easy)	
		Questions per module						
Widgets	Qt wx	Q10	Q15	Q17	Q19			Q19(Qt)
Database	Qt wx	Q1	Q13					
Main Window	Qt wx	Q6	Q14					Q6(wx)
Graphics	Qt wx	Q2	Q9			Q20		Q9(Qt)
IO	Qt wx	Q3				Q18	Q21	
Events	Qt wx	Q4	Q12					
Model View Document View	Qt wx	Q5	Q16					Q16(wx)
Layout	Qt wx	Q7	Q11				Q22	

Table 6.4. Layout and system characteristics. The number of edge crossings and edge bends for each layout. The number of classes and relationships is the same across both layouts.

Modules	System	Number of classes	Number of Relationships	Number of Crossings		Number of Bends	
				Layout Type		Layout Type	
				multi	ortho	multi	ortho
Widgets	Qt	23	25	0	0	27	25
	wx	23	26	0	0	14	36
Database	Qt	18	20	1	0	30	38
	wx	18	20	1	1	15	40
Main Window	Qt	17	18	0	0	16	20
	wx	17	17	0	0	11	26
Graphics	Qt	19	20	0	1	23	26
	wx	19	20	0	0	14	33
IO	Qt	18	17	0	0	14	26
	wx	18	19	0	0	15	17
Events	Qt	18	22	0	0	19	16
	wx	18	19	0	0	12	19
Model View Document View	Qt	22	23	0	0	16	23
	wx	22	27	2	2	28	33
Layout	Qt	24	27	1	0	20	28
	wx	24	24	0	0	13	22

6.3.4 Data Collection

The study was conducted online at the subject's place of choice. Four online questionnaires were used: background questionnaire, *Qt* questionnaire, *wxWidgets* questionnaire and, the debriefing questionnaire. The background questionnaire gathered information about the subjects including a self-assessment of their programming and design abilities. The second and third questionnaires formed the first and second run of the experiment for each system. Each question in each task category was scored on a scale from 0 to 1 depending on the number of correct answers. A sum of all the scores

represents the accuracy. The speed i.e., time taken to complete each question was also recorded discreetly via a timing mechanism implemented in the software. The subjects were instructed to answer the questions quickly to the best of their ability. Besides the accuracy and speed, we also collected a confidence level (Likert scale 1=very confident to 5=not confident) of the subject's answer for each question. The debriefing questionnaire collected data after each run, about task clarity, realism, understandability, difficulty, and whether stereotypes helped in answering the questions.

6.3.5 Study Subjects

A total of forty-five students (twenty-three design experts, twenty-two design novices) participated from two universities. Two of these participants were from industry (experts in design and programming). The subjects were randomly assigned to one of two groups. The subjects had varied programming and design expertise. Both graduate and undergraduate students participated. The subjects were classified into experts or novices based on their grades in the course they were enrolled in. The subjects were informed that the purpose of the study was to understand how people interpret class diagrams (not their UML proficiency). They were also instructed to answer the questions from the viewpoint of a maintainer trying to understand the system. Most of the subjects were not familiar with the design of either system.

6.3.6 Running the Experiment

A couple of days before the experiment, the subjects were asked to go through a class diagram tutorial. A short description of class stereotypes and their graphically

representation was given. They were also informed of the colors used to differentiate between different class stereotypes. The tutorial was optional; however, all subjects with an exception of a few participated in the tutorial and stated that it helped them refresh their knowledge of class diagrams. After the tutorial, the study was completed in two runs.

During each run, subjects were first presented with a set of instructions and a one screen description (on a 17" monitor with the resolution at 1280*1024) of the system (*Qt* or *wxWidgets*). We recommended that the subjects take the study on a 17" monitor with the appropriate resolution to view most of the diagram on one screen with minimal scrolling. Most of the questions were multiple-choice. After each question, they were prompted to enter a confidence rating for the question they just answered. There was no time limit set. The only requirement was they finish each run in one sitting. The following information was presented for each question: the question statement, answer choices and a class diagram for *Qt* (or *wxWidgets*) in one of two possible layouts. The subjects were asked to choose the answer for the question with respect to the class diagram. After completing all the tasks, subjects were presented with a debriefing questionnaire.

6.4 Experimental Results and Analyses

This section presents the results of the comprehension and preference parts of the experiment. Descriptive statistics are given in Table 6.5 along with *p-values* in Table 6.6. Box plots for *accuracy* and *speed* are given in Figure 6.1. We report on the effect of layout on accuracy and speed in all task categories as well as in each of the six task categories. Effect size is also reported using Cohen's *d* for accuracy and speed to facilitate easy comparison to other studies. Since this is a within-subjects study, we use the paired Wilcoxon non-parametric test to determine significance of the results.

Table 6.5. Descriptive statistics for *accuracy* and *speed* in each task category for each system.

Layout	System	Tasks	Mean Accuracy (Speed)	Standard Dev. Accuracy (Speed)
G r o u p A (N = 2 7)				
Orthogonal Layout	wx-Widgets	All	10.354 (1670.74)	4.507 (534.945)
		Reading	3.104 (174.148)	1.110 (68.201)
		Overview	3.524 (614.778)	1.970 (218.547)
		Impact Analysis	1.25 (306.037)	0.943 (165.435)
		Bug Fix	0.333 (134.852)	0.339 (55.851)
		Feature Add.	1.296 (272.556)	0.948 (104.677)
		Refactoring	0.845 (168.370)	0.508 (128.093)
Multi-cluster Layout	Qt	All	12.759 (1344.667)	4.818 (754.180)
		Reading	3.518 (139.741)	0.726 (59.791)
		Overview	4.827 (526.333)	2.651 (274.742)
		Impact Analysis	1.487 (244.296)	0.674 (173.291)
		Bug Fix	0.481 (67.556)	0.353 (71.606)
		Feature Add.	1.530 (212.556)	0.902 (157.966)
		Refactoring	0.913 (154.185)	0.553 (119.803)
G r o u p B (N = 1 8)				
Orthogonal Layout	Qt	All	9.619 (1620.778)	4.070 (676.805)
		Reading	3.111 (132.500)	1.118 (71.837)
		Overview	3.332 (675.278)	2.159 (313.818)
		Impact Analysis	1.175 (262.000)	0.549 (184.309)
		Bug Fix	0.027 (76.889)	0.117 (53.619)
		Feature Add.	0.888 (267.833)	0.626 (135.577)
		Refactoring	1.083 (206.278)	0.580 (145.070)
Multi-cluster Layout	wx-Widgets	All	12.05 (1264.111)	4.722 (840.271)
		Reading	3.5 (129.833)	0.923 (84.025)
		Overview	4.407 (542.778)	2.436 (418.339)
		Impact Analysis	1.569 (170.222)	0.910 (141.417)
		Bug Fix	0.527 (68.778)	0.468 (76.069)
		Feature Add.	1.111 (171.222)	0.824 (141.976)
		Refactoring	0.935 (181.278)	0.533 (156.820)

Table 6.6. Task categories study: 1-tailed p -values for the paired Wilcoxon test ($\alpha=0.05$). Cohen's d denotes the effect size: 0.2(small), 0.5 (medium), ≥ 0.8 (large). Alternative hypotheses for the 1-tailed tests indicate the following directionality: $\mu(\text{Accuracy}_{\text{ortho}}) < \mu(\text{Accuracy}_{\text{multi}})$ and $\mu(\text{Speed}_{\text{ortho}}) > \mu(\text{Speed}_{\text{multi}})$. * indicates significance.

Tasks	Cohen's d for Accuracy (Speed)		1-tailed p(Wilcoxon) Accuracy (Speed)	
G r o u p A (N = 2 7)				
All	0.51	(0.1)	0.000036 *	(0.001 *)
Reading	0.44	(0.5)	0.035 *	(0.048 *)
Overview	0.55	(0.35)	0.00091 *	(0.019 *)
Impact Analysis	0.28	(0.3)	0.0295 *	(0.025 *)
Bug Fix	0.427	(1.04)	0.01 *	(0.0003 *)
Feature Addition	0.252	(0.4)	0.07	(0.007 *)
Refactoring	0.12	(0.1)	0.233	(0.06)
G r o u p B (N = 1 8)				
All	0.55	(0.46)	0.00005 *	(0.006 *)
Reading	0.37	(0.03)	0.0058 *	(0.4)
Overview	0.46	(0.35)	0.0053 *	(0.03 *)
Impact Analysis	0.52	(0.50)	0.037 *	(0.007 *)
Bug Fix	1.46	(0.12)	0.00048 *	(0.16)
Feature Addition	0.30	(0.69)	0.09	(0.00001*)
Refactoring	0.26	(0.1)	0.874	(0.187)

6.4.1 Effect of Layout on Accuracy

The alternative hypothesis to H_{0a} (in Section 6.3.1) is that the multi-cluster layout performs better (has a higher accuracy) than the orthogonal layout. We can reject the null hypothesis H_{0a} considering all tasks in both Groups A and B (p -value=0.000036 and 0.00005 respectively). The effect size is medium (0.51 and 0.55) for both groups. However, after performing a finer grained significance test on each task category, we cannot reject H_{0a} for the feature addition tasks (Group A p -value=0.07, Group B p -value=0.09) and the refactoring tasks (Group A p -value=0.233, Group B p -value=0.874) in either group. The effect sizes for these two task categories (feature addition and refactoring) are small (Cohen's $d \sim 0.2$) compared to the rest of the tasks that were found

to improve significantly (medium effect) with the multi-cluster layout. The only thing we can conclude from this, is that there was no significant difference (with a small effect) in layout for these two task categories.

Figure 6.1. Descriptive Statistics for accuracy and time across groups for the task categories study

This outcome is caused by the specifics of the questions involved in these task categories. The questions did not take advantage of information in clusters causing both the layouts to have no significant difference. Since refactoring tasks focused on a small very specific portion of the hierarchy (such as a pull up field/ collapse hierarchy refactoring), layout did not have much effect. One of the three feature addition tasks (Q19) was much more difficult than the other two. This might have also affected the result. A question by question analysis between layouts is left as a future exercise.

Over all 45 subjects, the Pearson correlation indicates a significant positive correlation between accuracy and programming and design skills reported by subjects in the background questionnaire ($p\text{-value}<0.0024$ $r_s=0.41$). We report this correlation to validate our subject categorization of experts and novices.

6.4.2 Effect of Layout on Speed

The alternative hypothesis to H_{0s} (in Section 6.3.1) is that the multi-cluster layout takes less time to complete a task compared to the orthogonal layout. This can be considered to be the effort required to complete a task. We can reject the null hypothesis H_{0s} for all tasks in both Groups A and B ($p\text{-value}=0.001$ and 0.006 respectively). See Figure 6.1 for distributions. The effect size is low (0.1) in Group A and close to medium (0.46) in Group B.

A finer grained analysis per task category shows no significance in speed between the two layouts for the refactoring tasks in Group A ($p\text{-value}=0.06$). The same is the case for Group B ($p\text{-value}=0.187$). This effect could be attributed to the same earlier fact that refactoring tasks were considered to be easy tasks focusing on a small portion of a

hierarchy. It is also important to note that the effect size is very small (~ 0.1) in both groups indicating a very small effect that both the layouts are not different. In addition to the refactoring tasks, we cannot reject H_{0s} for the reading and bug fix tasks for Group B, albeit with a very small effect size (0.03 for reading and 0.12 for the bug fix task).

We also notice a high effect in accuracy (1.46) for the bug fix task in Group B, but a low effect in terms of time. From this we observe that even though the time taken was not significantly different, the accuracy for the multi-cluster layout was significantly improved. The same observation is detected in the reading task category in Group B. In Group A, the bug fix task had a large effect (1.04) in speed: both accuracy and speed were significantly improved for the multi-cluster layout.

The only task category that we could not reject H_{0a} and H_{0s} was for the refactoring tasks for both groups (with a small effect). The refactoring tasks were considered to be easy and did not take advantage of layout as much as the other tasks, since they focused on a very specific area of the diagram and did not involve searching for related classes or relationships. For the difficult and challenging task categories, speed of task completion was significantly lower for the multi-cluster layouts with a medium effect on average. A medium effect is considered to be practically significant.

6.4.3 Secondary Factor Interactions

The third null hypothesis H_{0e} (in Section 6.3.1) seeks to determine if the secondary factors such as design experience or the systems used, interacts with the layout while performing the tasks. We perform a secondary factor analysis on the whole data set (both groups $N=45$). Here, we satisfy the conditions of the *ANOVA* significance test

(normality was measured using *Shapiro-Wilk*). We conduct a 2-way *ANOVA* between layout and design experience and between layout and system.

Results indicate that design experience does not significantly interact with the layout type (orthogonal or multi-cluster) to have an effect on task comprehension accuracy or speed. We cannot reject H_{0e} . The same is true for the interaction between layout and system used. This validates the fact that the systems chosen were comparable in nature. However, *ANOVA* did report a direct significant effect due to the experience factor alone for both accuracy ($p\text{-value}=0.02$) and speed ($p\text{-value}<0.0001$), indicating a significant difference in the way experts and novices comprehend diagrams. During this analysis we noticed that novices tend to benefit more from the multi-cluster layout and took much less time to complete the tasks. See interaction plot in Figure 6.2. A 3-way *ANOVA* between layout, design experience and system used found no additional interactions.

To summarize, we revisit the research questions we posed earlier in Section 6.1. With respect to RQ6.1, we find the difficult and challenging task categories to benefit most from the stereotyped multi-cluster layouts. With respect to RQ6.2, experts gave more correct answers in the multi-cluster layout than novices and there was a slightly bigger difference in average accuracy between the two layouts for experts compared to novices. However, in terms of time, novices had a bigger difference between the two layouts as shown in Figure 6.2. This states that the multi-cluster layout helped novices answer the question much quicker compared to experts. We also find that novices spent less time on average compared to experts. One possible explanation for this could be due

to the fact that the experts approached the study in a more professional manner and took the study more seriously.

Figure 6.2. Interaction between layout and design experience with respect to time for the task categories study

6.4.4 Confidence Level

The confidence level ratings for each question were used to determine if a correlation exists between confidence reported by subjects and accuracy of tasks. The *Spearman* rank correlation between the confidence level for all questions compared to the accuracy of all the questions showed a positive correlation. (Group A wx: $r_s=0.5, p\text{-value}=0.0036$, Group A Qt: $r_s=0.55, p\text{-value}=0.0014$, Group B wx: $r_s=0.78, p\text{-value}<0.0001$, Group B Qt: $r_s = 0.67, p\text{-value} = 0.0011$). This states that subjects were able to self-assess their responses to the questions in a majority of the questions. In general, a majority of the task categories had consistently higher confidence ratings for

the multi-cluster layout compared to orthogonal layout. In cases where there was no difference in layout, the confidence level was also similar in both layouts.

6.4.5 Debriefing Questionnaire

The debriefing questionnaire asked the subjects to rate the difficulty level of *Qt* and *wxWidgets*. The *Pearson's* correlation (N=45) between the difficulty level of each system shows a positive correlation ($r_s=0.59$, $p\text{-value}<0.0001$). This shows that the subjects found the difficulty level to be around the same. A *Mann-Whitney* 1-tailed test on all the responses (N=45) in the debriefing questionnaire after each run in each group found no significant differences between the two runs in terms of difficulty, clarity, time needed and task realism. In both runs, stereotypes were found helpful in answering the questions based on a Likert scale rating.

6.4.6 Preference Ratings on Two Comprehension Tasks

After completing the comprehension part of the experiment, subjects were asked to rate the multi-cluster and orthogonal layouts for two questions (See Table 6.3) based on two criteria: comprehension and aesthetics. We analyze these preferences using the unpaired *Mann-Whitney* test (since the ratings are on an ordinal scale) for each question (N=45). See Table 6.7 for the results.

The aesthetics were comparable i.e., no significant difference was found between the two layouts in the *Qt* system. In terms of comprehensibility of the layouts, the multi-cluster layout was found better for the Overview task-Q9. The feature addition task's comprehension preference ratings did not show any significance. This question (Q19) in

particular was much harder and this resulted in a low rating for both layouts. Subjects were not sure if they could answer this correctly and as a result tended to rate both layouts equally badly. This matches our analysis for this task category in Section 6.4.1. In *wxWidgets*, the multi-cluster layout was significantly better in both comprehension and aesthetic ratings. This was probably due to the fact that the orthogonal layout had a larger number of bends compared to the multi-cluster layout (See Table 6.4). In general, the preference ratings tend to mimic the comprehension results presented above.

Table 6.7. Mann-Whitney results (1-tailed) of preference ratings for 2 questions in each system for the task categories study

Question	Module	Task Category	p(Mann-W) Comp multi > ortho	p(Mann-W) Aesthetic multi > ortho
Q9 - Qt	Graphics	Overview	0.01 *	0.20
Q19 - Qt	Widgets	Feature Addition	0.133	0.98
Q6 - wx	Main Window	Overview	0.0004 *	0.00001 *
Q16 - wx	DocumentView	Impact Analysis	0.005 *	0.014 *

Many subjects commented that they considered the closeness of related classes and relationships to make their choice. Other comments included the ease of tracing relationships as important. Short lines were preferred over long ones. They also preferred a less ‘cluttered’ look. One subject pointed out that their rating was based on the ability to break the diagram into smaller parts easily. This is precisely what the multi-cluster layout does.

6.5 Threats to Validity

Since the diagrams were manually engineered, there is the possibility that the type of task might favor a certain layout such that relevant classes for the task are closer in the diagram. The questions in this study were designed in a way that minimizes such situations. The answers to certain tasks were sometimes found not in just one cluster but a combination of clusters. Also, care was taken to ensure aesthetic criteria in all diagrams. The multi-cluster layouts were not drawn to match the tasks, rather they represent certain features in the system.

Being a within-subjects study, learning effects were avoided by using two systems and conducting the study on different days. This experiment was part of a subject's grade in a course. They received credit for participating in the whole experiment not on actual performance. The students we used as subjects had varied experience in programming and design. The experts were comparable to mid-level to senior developers. The subject systems used were real applications. The questions in each task were derived from the repository information available for both systems. Overall, the subjects agreed that the questions were clear, realistic and the information in the class diagrams was understandable.

All conditions for the statistical tests were tested to ensure conclusion validity. We use the non-parametric Wilcoxon test to determine significance due to a small sample size and to perform paired analyses. *ANOVA* and the unpaired *Mann-Whitney* test (for preference ratings) were used where appropriate.

6.6 Summary

The controlled experiment empirically validates two layouts in six categories of software tasks using two open source systems. To our knowledge, this is the first attempt at conducting such an analysis on class diagram layouts. The multi-cluster layout achieves a higher level of accuracy and takes less time than the orthogonal layout for a majority of the task categories including difficult and challenging tasks. In addition, design novices complete the tasks much faster than experts for the multi-cluster layout compared to the orthogonal layout. A preference rating also detects that subjects preferred the multi-cluster layout over the orthogonal one when asked to rate the two diagrams with respect to a task.

CHAPTER 7

A Questionnaire-based Study on Detecting the Role of Design Patterns

A controlled experiment investigating the effect layout has on how students identify design pattern roles in UML class diagrams is presented. Two layout schemes, multi-cluster and orthogonal, are compared with respect to three open source systems and four design patterns. Seventeen students were asked a series of eight design pattern role detection (comprehension) questions for each layout, followed by eight preference rating questions. Results indicate a significant improvement in role detection accuracy with the multi-cluster layout for the strategy pattern and a significant improvement in detection time with the multi-cluster layout for all four patterns. Preference ratings significantly favored the multi-cluster layout for pattern role detection ease. These results can be used to help improve the teaching of design patterns.

7.1 Design Patterns and Motivation

Design patterns are reusable solutions to common software engineering problems. Class diagrams are typically introduced in software engineering courses during the introduction of object oriented concepts. Design patterns are also taught using (UML-like) class diagram templates as described in Gamma et al. [Gamma, Helm, Johnson, Vlissides 1995] and many instructors present this material in actual UML. An example of the Strategy pattern template is shown in Figure 7.1. The

aggregation and generalization relationships are present in this pattern. There may be one or more concrete strategies available.

Figure 7.1. An Example of the Strategy pattern template. The class Stat plays the role of Context, SortImpl plays the role of Strategy and SortBubble and SortShell play the roles of Concrete Strategies

These visual representations of patterns are typically shown in isolation with only the relevant classes involved in the design pattern. However, in real software, classes that represent a design pattern also interrelate with other classes in the system. During maintenance, a programmer is not given the set of design pattern classes in isolation rather; he/she needs to identify relevant classes or relationships with respect to the task at hand.

In prior work, [Andriyevska, Dragan, Simoes, Maletic 2005; Sharif, Maletic 2009a; b], we assess the usefulness of class stereotypes (control, boundary, and entity) in class diagram layouts with respect to specific software tasks. Along with general aesthetics [Eichelberger 2003; Eiglsperger, Kaufmann, Siebenhaller 2003; Gutwenger et al. 2003b; Purchase, Alder, Carrington 2002; Purchase, McGill,

Colpoys, Carrington 2001], the layouts are primarily organized into meaningful clusters, where each cluster consists of control, boundary, and entity classes working together. The results indicate that clustered layouts play a significant role in comprehension accuracy of tasks as well as in the time taken to complete a task. This study seeks to empirically determine if clustered layouts (and more generally if different layouts) help in detecting roles for design patterns in class diagrams. Students are usually not taught layout techniques while learning class diagram UML notation. If layout improves comprehension and time taken to perform a task, students would benefit from learning about layout while UML notation and design patterns are taught. Students would be made aware of the fact that aesthetics are only one part of making a diagram easy to comprehend.

The next section introduces the types of layouts used in this study. The experimental design is outlined next followed by the results. Related work on how design patterns are taught is presented in Section 7.6 followed by summary and future work.

7.2 Class Diagram Layouts

In this study, we refer to a class diagram as the (sub)set of classes and relationships visually represented on a medium such as the screen or paper with a particular layout scheme applied. This study uses two layout schemes: orthogonal and multi-cluster to assess design pattern role detection. Class stereotypes are visually presented in the diagram using textual annotations above the class name as

well as in color: red for control classes, blue for boundary classes and green for entity classes. The layouts are described in Chapter 3.

7.3 Experimental Design

The experiment seeks to *analyze* two class diagram layouts *for the purpose of* evaluating their usefulness in design pattern role detection *with respect to* effectiveness (accuracy) and efficiency (time) *from the point of view of* the researcher *in the context of* students at Kent State University.

The research questions this paper attempts to address are:

- RQ7.1: Do clustered layouts improve design pattern role detection accuracy and time in UML class diagrams?
- RQ7.2: Which design patterns benefit the most from the clustered layouts?
- RQ7.3: Do clustered layouts help design experts and novices in the same way?
- RQ7.4: Which layout is preferred for solving the role detection task in a design pattern?

The detailed null hypotheses are given below. The alternative hypotheses are 1-tailed predicting the multi-cluster layout performs better.

H₁: There is no significant difference in design pattern role detection *accuracy* between class diagrams in the orthogonal layout and the multi-cluster layout.

H₂: There is no significant difference in design pattern role detection *time* between class diagrams in the orthogonal layout and the multi-cluster layout.

H₃: There is no significant difference in preference ratings between the multi-cluster layout and the orthogonal layout with respect to *pattern role detection ease*.

H₄: There is no significant difference in preference ratings between the multi-cluster layout and the orthogonal layout with respect to *aesthetics*.

H₅: Experience does not interact with layout to have an effect on *accuracy*.

H₆: Experience does not interact with layout to have an effect on *time*.

A within-subjects design was used, where each subject was tested on the role detection of four design patterns in both layouts albeit in different systems to alleviate any learning effects. The overview of the experiment is shown in Table 7.1. Refer to Figure 7.2 for an example of the two layouts in the Qt system showing the Strategy pattern.

The study was conducted online at the subject's place of choice. The background questionnaire gathered information about the subjects. The main questionnaire contained the role detection (comprehension) and preference parts of the study. The post questionnaire collected data about question difficulty, clarity, and criteria the subjects used to determine their answers.

Table 7.1. Experiment overview for detecting design pattern roles study

Goal	Study the effect of two layout schemes for class diagrams in the context of identifying classes and their roles in design patterns.
Main Factor (Independent var.)	Class diagram layouts with two treatments: orthogonal layout, multi-cluster layout
Dependent variables	Accuracy, time, relevance, confidence level

Figure 7.2. The Strategy pattern in *Qt* shown in both layouts. The roles of context, strategy and concrete strategy are performed by `QXmlContentHandler`, `QXMLReader`, and `QXMLSimpleReader` respectively.

Before starting the main questionnaire, the subjects were asked to complete a refresher class diagram tutorial and a design pattern tutorial.

7.3.1 Subjects

A total of 17 (10 undergrads, 7 grads) students participated in the study. The subjects were informed that the purpose of the study was to understand how software engineers interpret design pattern roles in class diagrams. They were also asked to answer the question as quickly as possible to avoid unnecessary timing delays. All subjects were first introduced to design patterns in academia. None of the students were taught a layout style while learning UML in the classroom.

7.3.2 Subject Systems And Design Pattern Selection

We chose three open-source systems: *JUnit*, *JHotDraw* and *Qt*, that are well designed and use design patterns. Refer to Table 7.2. A total of 112 unique classes were used in this study. Diagrams in two different layouts were then manually engineered in a UML drawing editor by inspecting the code and online documentation for entity, boundary and control classes that work together towards a specific feature or functional requirement. While drawing the multi-cluster layouts, classes that participate in a design pattern were shown in one cluster and drawn to match the standard template form as shown by Gamma et al. [Gamma, Helm, Johnson, Vlissides 1995]. Due to time, subject number, and subject fatigue constraints only four of the most commonly taught and used patterns were chosen for our study (See Table 7.2).

Table 7.2. Overview of subject systems, classes and design patterns used

System	Lang.	KLOC	Version.	# Classes used	Design patterns used
JUnit	Java	9	4.6	27	Composite, Observer
JHotDraw	Java	15	5.1	49	Composite, Observer, Strategy, Singleton
Qt	C++	729	4.3.3	36	Strategy, Singleton

Table 7.3. Role detection questions in four patterns across three systems for the questionnaire-based study

ID	Design Pattern	System	Layout	# crossings	# clusters	# classes	# relations
Q1	Composite	JUnit	Multi-cluster	1	4	17	21
Q2	Observer	JUnit	Multi-cluster	1	4	20	28
Q3	Strategy	Qt	Orthogonal	0	-	17	18
Q4	Singleton	Qt	Orthogonal	0	-	19	19
Q5	Composite	JHotDraw	Orthogonal	0	-	17	19
Q6	Observer	JHotDraw	Orthogonal	1	-	20	24
Q7	Strategy	JHotDraw	Multi-cluster	1	4	17	20
Q8	Singleton	JHotDraw	Multi-cluster	0	4	19	19

7.3.3 Comprehension and Preference Questions

The study was split into two parts: comprehension and preference. The comprehension part consists of eight role detection questions (See Table 7.3). Care was taken to keep the number of crossings between layouts consistent. Each question was accompanied by a diagram in a particular layout. For example, in Q1, the subjects were told that the diagram contains the Composite pattern and were asked to assign a class to the Composite, Component and Leaf roles. Note that the subject was not asked to identify a particular design pattern contained in a diagram,

which is a much harder problem. During the analysis we compare Q1 with Q5, Q2 with Q6, Q3 with Q7 and Q4 with Q8. Each question was given a score based on the accuracy of role assignment. The total score for each question was the sum of each role assignment score. Time taken for each question was also recorded. After each question, the subject was asked to rate on a Likert scale (1 to 5) their level of confidence, ease of role detection and the intuitiveness of the class names for the answer they just provided. The preference part of the study asked the subjects to rate eight diagram pairs in each layout in terms of aesthetics and design pattern role detection ease. In the preference part, we compare two layouts of the same system. For example, the first question asks to rate the two layouts for the Composite design pattern in *JUnit*. These questions were scored on a Likert scale (1 to 5).

Table 7.4. Questionnaire based study: 1-tailed Wilcoxon p -values (alpha=0.05) for accuracy, time, and relevance for each design pattern's role detection. Directionality implies that the multi-cluster layout performs better. Cohen's d denotes the effect size: 0.2(small), 0.5 (medium), ≥ 0.8 (large). * indicates significance.

Role Detection Accuracy				Total Accuracy (Cohen's d)	Relevance (Cohen's d)	Time (Cohen's d)
Composite Pattern				0.001 * (0.5)		
Roles	<i>Composite</i> 0.813	<i>Component</i> 0.656	<i>Leaf</i> 1.0	0.766 (0.2)	0.344 (0.05)	
Observer Pattern				0.037 * (0.3)		
Roles	<i>Subject</i> 0.5	<i>Observer</i> 0.5		0.344 (0.1)	0.344 (0.1)	
Strategy Pattern				0.002 * (0.7)		
Roles	<i>Concrete Strategy</i> 0.109	<i>Context</i> 0.109	<i>Strategy</i> 0.035 *	0.02 * (0.6)	0.001 * (0.8)	
Singleton Pattern				0.007 * (0.4)		
Role	<i>Singleton</i> 0.5			0.5 (0.08)	0.5 (0.08)	

7.4 Experimental Results

This section analyzes and reports results of the comprehension and preference parts of the study with respect to our hypotheses.

7.4.1 Accuracy and Time

Since our experiment followed the within-subjects design, we used the paired Wilcoxon test to determine significance of the results. The results of role detection accuracy and time are shown in Table 7.4 with the box plots shown in Figure 7.3. In addition to the total accuracy, we also measure the relevant classes detected in a

pattern by the subject. For instance, if the subject identifies all the classes in the Observer pattern but reverses the roles for Subject and Observer, the total accuracy would be 0 however, the relevance would be 2. From the results we observe that the relevance measure tends to follow the total accuracy measure for all patterns tested, which means that very rarely were roles mixed up.

Results are now discussed with respect to the hypotheses presented in Section 7.3. The alternative hypothesis to H_1 states that the multi-cluster layout correctly detects significantly more roles in design patterns than the orthogonal layout. We can only accept the alternative hypothesis in the case of the Strategy pattern that performs significantly better in total accuracy ($p\text{-value}=0.02$), the Strategy role ($p\text{-value}=0.035$) and relevant classes chosen ($p\text{-value}=0.001$). The significance of the relevant classes is much higher than the total accuracy indicating that the concrete strategy and context roles were mixed up. We can accept the alternative hypothesis to H_2 (multi-cluster takes less time) for all four design pattern role detection tasks. See last column in Table 7.4 for $p\text{-values}$. Effect sizes, calculated using Cohen's d , tell us the extent of significant findings. With respect to time, the effect sizes for the Composite, Strategy and Singleton are medium or higher which are considered to be practically significant. With respect to total accuracy, we have a small effect for all but the Strategy pattern (Cohen's $d = 0.6$). This indicates that the layout was practically significant for the role detection tasks in the Strategy pattern.

Figure 7.3. Accuracy and time box plots for patterns across layouts for the questionnaire-based study

7.4.2 Secondary Factor Interactions

This section addresses our third research question (RQ7.3) presented in Section 7.3. The subjects consisted of seven experts mainly graduate students and ten novices

(undergraduates) in design. This analysis seeks to determine if experience played a role in the accuracy, time and relevance dependent variables studied. Even though the sample size is low, *ANOVA* may be used to determine interaction effects of other factors as stated in [Wohlin et al. 2000] even though the distributions are not normal due to the fact that *ANOVA* is very robust. In our case, the distributions for accuracy and time in all four patterns are not normally distributed. Even though the normality constraint is violated, we still perform the *ANOVA* test due the reason mentioned above. See Table 7.6 for *ANOVA* results.

ANOVA did not find any significant interaction effects between experience and layout used. Hence we cannot reject the null hypotheses H_5 and H_6 . However, experience by itself was significant in the Composite, Observer and Strategy patterns for the accuracy dependent variable. Layout was significant in the Strategy pattern for accuracy and time variables. These results match the results of the Wilcoxon test presented in the earlier section.

Table 7.5. Results of the *ANOVA* test with the Experience factor. p-values for Accuracy (Time) are mentioned for each pattern. * indicates significance.

	Composite	Observer	Strategy	Singleton
Experience	0.0001 * (0.240)	0.035 * (0.128)	0.005 * (0.137)	0.07 (0.477)
Layout	0.585 (0.142)	0.695 (0.356)	0.049 * (0.024 *)	0.904 (0.276)
Experience * Layout	0.199 (0.983)	0.695 (0.525)	0.815 (0.158)	0.472 (0.914)

Next, we inspect the interaction plots for each pattern in each dependent variable: accuracy, time and relevant classes to get an idea of the type of interaction that could be

possible albeit not significant. The interaction plots are simple graphs that plot the means of the dependent variable for each level of the secondary factor, in this case experience. Experience has two levels: experts and novices. Parallel lines indicate no interaction, whereas non parallel lines indicate that interaction may be present.

The interaction of experience (secondary variable) with the layouts used are shown in Figure 7.4. We can make the following observations from the interactions.

- Experts consistently scored higher than novices in all four design patterns.
- Experts took more time than novices for all four design patterns. Novices are not as serious as experts when participating in experiments.
- Experts found more relevant classes than novices.
- The multi-cluster layout reduces the gap between experts and novices (except for the Composite pattern) for accuracy performance.
- The multi-cluster layout reduces the gap between experts and novices (except for Composite and Singleton patterns) for time performance. This can be seen by the non-parallel lines in b) and c) for the Time dependent variable.

a) Composite

b) Observer

c) Strategy

d) Singleton

Figure 7.4. Interaction between Layout and Experience for the four patterns for each dependent variable i.e., accuracy, time and relevant classes

7.4.3 Preference Ratings

The *Mann-Whitney* non-parametric test is used to determine significance since we are not dealing with paired values (See Table 7.6). Each layout was rated individually on a Likert scale(1-5). We can reject the null hypothesis H_3 for all but two cases as shown. The multi-cluster was significantly preferred over the orthogonal layout with respect to pattern detection ease (except in the Composite case in *JHotDraw* and the Singleton case in *Qt*). The only cases where the multi-cluster layout was preferred in terms of aesthetics over the orthogonal layout was for the Observer pattern in *JHotDraw*, Strategy pattern in *Qt* and the Singleton pattern in *JHotDraw*. Overall, we cannot reject the null hypothesis H_4 indicating no difference in aesthetics between the two layouts.

Table 7.6. Questionnaire-based study: *Mann-Whitney p-values* (1-tailed: multi-cluster > orthogonal, alpha=0.05) for preference ratings. * indicates significance

Pattern	System	Aesthetics	Pattern Detection Ease	Role
Composite	JUnit	0.137	0.0001 *	
	JHotDraw	0.270	0.393	
Observer	JUnit	0.527	0.036 *	
	JHotDraw	0.02 *	0.007 *	
Strategy	Qt	0.05 *	0.04 *	
	JHotDraw	0.9	0.03 *	
Singleton	Qt	0.793	0.393	
	JHotDraw	0.05 *	0.046 *	

7.4.4 Observations

The post-questionnaire analysis indicates all subjects found the difficulty level between *JUnit*, *JHotDraw* and *Qt* comparable. The subjects rated the Strategy and Observer pattern roles as the most difficult to detect. When asked about the criteria

used to detect roles, subjects stated that they mainly used the relationships and spatial layout. Attributes and method information were used by 50% of subjects, mostly graduate students. The confidence level for Composite, Observer and Singleton were significantly positively correlated with the accuracy of role detection for both layouts. Strategy pattern roles were harder to detect and were given a lower confidence rating by subjects. The Strategy pattern consists of an aggregation and a hierarchical relationship. Since the aggregation is not within the hierarchy, it tends to be placed further away in the orthogonal layout. The multi-cluster layout recognizes cohesive clusters including aggregations and not just hierarchical groupings. This causes the Strategy pattern to have a higher accuracy with less time spent in role detection for the multi-cluster layout. The Composite pattern is similar but the aggregation is within the same hierarchy that makes it easier to detect. This paper shows that layout can have a significant effect in the amount of time a subject takes to detect roles. Detecting roles however, is a very small part of the big picture. In reality, a maintainer would have to not only identify roles or design patterns but also find how they relate to solve a particular task. If layout reduces the time it takes to read and understand a diagram, more time can be spent on the actual maintenance task at hand.

7.5 Threats to Validity

With respect to internal validity, we address learning effects by using a different system for the role detection of the same pattern for the two layouts. The subjects were sufficiently motivated to do well as this was part of their grade. Each

role detection question and its corresponding diagram did not contain any other patterns i.e., Strategy detection diagrams did not contain Composite patterns to avoid any confounding factors. To ensure construct validity, we designed our experiment with real software containing design patterns, not an example toy application. Subjects were not aware of the different layouts used, the hypotheses of the study, or the design of subject systems used. The sample we chose is representative of our target population i.e., students learning design patterns, which addresses external validity. To maintain conclusion validity, we use the appropriate non-parametric tests. Caveats are mentioned where ANOVA is used even though constraints might be violated.

7.6 Prior Work on Design Pattern Education

Jeanmart et al. [Jeanmart, Guéhéneuc, Sahraoui, Habra 2009] studied the effect of the visitor pattern on program comprehension and maintenance using an eye tracker. One of the results was that the visitor pattern's canonical form as presented in design pattern books required less effort from developers in maintenance tasks. Our results support this for the role detection tasks of design patterns. More details about this study are presented in Section 2.4.2.2 on page 56.

The rest of this section discusses how educators teach design patterns in the classroom setting. Alphonse et al. [Alphonse, Caspersen, Decker 2007] describe the challenges educators face in teaching design patterns and present three killer examples from the killer examples workshop. Several other works propose various teaching methods for design patterns such as frameworks [Christensen 2004; Della,

Clark 2000], multimedia [Dukovich, Janzen 2009], game programming [Gómez-Martin, Jiménez-Díaz, Arroyo 2009], and musical composition [Hamer 2004]. Warren [Warren 2005] proposes teaching design patterns to students by making them extend the *JUnit* system using design patterns. Weiss [Weiss 2005] proposes teaching design patterns by stealth without introducing the theory of design patterns first. Our approach of using layout is seen as complementary to the various design pattern teaching approaches presented above. It can be used in conjunction with the teaching methods presented.

7.7 Summary

The study validates the premise that layout has an effect on how students identify design pattern roles in class diagrams. The results of the study indicate a significant improvement in time taken to identify roles in four patterns with the multi-cluster layout. The Strategy pattern benefits the most from the multi-cluster layout with respect to correctly identifying roles. No significance difference was found with respect to accuracy of role detection between layouts for the Composite, Observer or Singleton patterns. The preference based ratings significantly favored the multi-cluster layout for role detection ease.

CHAPTER 8

An Eye-tracking Study on Detecting the Role of Design Patterns

This chapter presents the results of an eye-tracking study that replicates the previous study in Chapter 7. Similarities and differences between the two studies are discussed. This chapter presents an added analysis of the eye-tracking data that is missing from the previous study. The purpose of replicating this study is to gather additional insight into the thought processes via eye gaze that are not usually available via questionnaire-based methods. We also used a different sample population (not just students) in this study.

8.1 Experimental Design

The experimental design is the same as the original study (Chapter 7) with a few exceptions. Refer to Section 7.3 on page 158 for details on the design. The null hypotheses of this study are given below.

$H_{0_{\text{Accuracy}}}$: There is no significant difference in design pattern role detection *accuracy* between class diagrams in the orthogonal layout and the multi-cluster layout.

$H_{0_{\text{Time}}}$: There is no significant difference in design pattern role detection *time* between class diagrams in the orthogonal layout and the multi-cluster layout.

$H_{0_{\text{VisualEffort}}}$: There is no significant difference in the *visual effort* required for design pattern role detection between class diagrams in the orthogonal layout and the multi-cluster layout.

Refer to Table 7.1 for the dependent variables used. In this study we have one additional dependent variable: visual effort, in addition to accuracy and time. Eye fixation counts and durations are used to calculate the visual effort. Unlike the previous study, this study does not conduct a preference based rating for each pattern. This study also does not report on the confidence level for the task. More details on the variables in this study are presented in Section 8.1.4.

This study is also run as a within-subjects design. A class diagram tutorial and design pattern tutorial was presented to the subjects a few days before the study as well as on the day of the study.

8.1.1 Eye-tracking Apparatus

The *Tobii 1750* eye tracker (www.tobii.com) is used for this study. It is a video-based remote eye tracker that uses two cameras to capture eye movements. The cameras are built into a 17 inch TFT-LCD hardware. The screen resolution was set to 1024 by 768. This eye tracker does not require the subject to wear any form of head gear, thereby emulating a subject's normal work environment. The frame rate (temporal resolution) at which sampling occurs is 50 Hz, latency is around 25-35 ms, and average accuracy is 0.5 degrees (approx. 15 pixels average error). The eye tracker compensates for head movement during the study i.e., the eyes do not have to be focused on the screen all the time.

The *ClearView* analysis software that comes with the eye tracker was setup as a double screen configuration. The first screen is used by the moderator to setup and run the study. The second screen is used by the study subjects to perform the tasks. This lets

the moderator get real time feedback of the eye tracking quality during the task. The Tobii eye tracker records eye gaze and audio/video recordings of the entire study session. The eye gaze data include timestamps, gaze positions, eye positions, pupil size, and validity codes.

8.1.2 Stimuli

Diagrams used in this study are referred to as stimuli. The subject systems and design patterns used are the same as the previous study discussed in Chapter 7. Refer to Section 7.3.2 for more information. The main difference is in the layouts presented to the user in each pattern. In this study, the layout was interchanged for each pattern. This was done in order to test the other untested balanced half of the previous study. See Table 8.1 and contrast this to Table 7.3. The only difference is in the fourth column, Layout.

An example of a stimulus shown to the subjects is given in Figure 8.1. The top left corner of the screen consists of the task definition.

Which two classes participate in the Observer design pattern?
Name the role (*Subject*, *Observer*) of each class.

Figure 8.1. Stimulus for detecting the role of Observer in the JUnit System (Q2)

Table 8.1. Role detection questions in four patterns across three systems for the eye-tracking study

ID	Design Pattern	System	Layout	# crossings	# clusters	# classes	# relations
Q1	Composite	JUnit	<i>Orthogonal</i>	1	4	17	21
Q2	Observer	JUnit	<i>Orthogonal</i>	1	4	20	28
Q3	Strategy	Qt	<i>Multi-cluster</i>	0	-	17	18
Q4	Singleton	Qt	<i>Multi-cluster</i>	0	-	19	19
Q5	Composite	JHotDraw	<i>Multi-cluster</i>	0	-	17	19
Q6	Observer	JHotDraw	<i>Multi-cluster</i>	1	-	20	24
Q7	Strategy	JHotDraw	<i>Orthogonal</i>	1	4	17	20
Q8	Singleton	JHotDraw	<i>Orthogonal</i>	0	4	19	19

8.1.3 Comprehension Task

This study consisted of eight comprehension questions, four for each pattern in two different layouts under two different systems to avoid learning effects. Given a class diagram, the task in each question was to assign roles to classes that participated in a specific design pattern. The subjects were informed of the existence of the design pattern and had to state the name of the class that participated in each of the roles. During the analysis we compare Q1 with Q5, Q2 with Q6, Q3 with Q7 and Q4 with Q8. Each question was given a score based on the accuracy of role assignment. The total score for each question was the sum of each role assignment score.

8.1.4 Study Variables

This section discusses the dependent variables used in this study.

- Accuracy: The sum of all the scores for each pattern's role assignment. This is an integer number between 0 and maximum number of roles in a pattern.

- Time: The amount of time required to detect the roles for each pattern. This is measured in seconds.
- Relevance: The same as accuracy but ignores the exact matchup of role assignment. The value of this variable may be greater than or equal to the Accuracy variable. When greater, it denotes the mismatch of roles in a pattern. When equal, no mismatch is detected.

The visual focus of the eyes on a particular location triggers certain mental processes in order to solve a given task [Just, Carpenter 1980]. Due to this correlation, visual attention can be used to study the cognitive effort in solving a task. In this study, we define four areas of interest (AOIs). See Figure 8.2 for the defined areas of interest.

- Entire Diagram (T). This involves all the classes and relationships in the diagram.
- Classes participating in the design pattern (DP). This involves only the classes involved in a particular design pattern.
- Design pattern cluster (DPCluster). This area of interest consists of the classes and relationships participating in the design pattern.
- Non-relevant classes (non-DP classes). These are classes not participating in the design pattern.

An area of interest is also defined for the task definition located on the top left corner of the stimulus. However, we ignore any fixations in this area in our analyses.

Figure 8.2. Stimulus with areas of interest overlaid using red rectangles. The top left shows the task definition AOI. Each class's bounding box is an AOI. A design pattern cluster AOI is shown in dotted lines and shaded yellow. The classes at both ends of this cluster form the design pattern classes AOI. All classes outside the design pattern cluster are part of the non-design pattern classes AOI.

Eye gaze data can then be analyzed within each AOI to determine the visual attention of subjects. In Figure 8.2, each of the class rectangles belong to an AOI. The TestResult and TestListener classes and the relationship between them form a design pattern cluster.

Visual Effort: Denotes the amount of effort needed to arrive at the answer. We gather the following data for the AOIs on the stimuli. Two main eye gaze data are eye

fixations and saccades. A *fixation* is the stabilization of the eyes on an object on the stimulus. *Saccades* are quick movements from fixation to fixation. The eye tracker was set to filter fixations within 20 pixels with duration of at least 40 ms. Visual effort is determined using each of the following measures.

- Fixation Count(FC_T): The total number of eye fixations on the entire stimulus.

$$FC_T = \sum_{a \in \{entire\ stimulus\}} f(a) ,$$

where $f(a)$ gives the fixation count of an area of interest.

- Fixation Rate on Design Pattern Classes (FR_{DP}): The total number of eye fixations on the design pattern classes with respect to all classes on the stimulus.
- Fixation Rate on the Design Pattern Cluster ($FR_{DPCluster}$): The total number of eye fixations on the design pattern classes with respect to all classes on the stimulus, including the relationships involved in the design pattern.
- Fixation Rate on Non-Design Pattern Classes (FR_{non-DP}): The total number of eye fixations on the classes not participating in the design pattern with respect to all classes on the stimulus. These are classes not relevant to the design pattern.

A higher fixation count and fixation rate indicates more effort needed by subjects to solve the task. These measures are characterized as follows.

$$FR_{DP} = \frac{\sum_{a \in \{DPclasses\}} f(a)}{\sum_{a \in \{DPclasses\} \cup \{non-DPclasses\}} f(a)} ,$$

$$FR_{DPCluster} = \frac{\sum_{a \in \{DPclasses\} \cup \{DPrelationships\}} f(a)}{FC_T},$$

$$FR_{nonDP} = \frac{\sum_{a \in \{non-DPclasses\}} f(a)}{\sum_{a \in \{DPclasses\} \cup \{non-DPclasses\}} f(a)},$$

where $f(a)$ gives the fixation count of an area of interest.

- Average Fixation Duration (AFD_T): The average length of time of all fixations in all classes on the stimuli.
- Average Fixation Duration on Design Pattern Classes (AFD_{DP}): The average length of time of all fixations in classes participating in the design pattern.
- Average Fixation Duration on the Design Pattern Cluster (AFD_{DPCluster}): The average length of time of all fixations in classes participating in the design pattern, including relationships between classes.
- Average Fixation Duration on Non-Design Pattern Classes (AFD_{nonDP}): The average length of time of all fixations in classes not participating in the design pattern.

The unit of measure is milliseconds. More time spent analyzing the stimuli indicates more visual effort. Each of the above measures is shown below.

$$AFD_T = \frac{\sum_{a \in \{entirestimulu\}} g(a)}{FC_T} ,$$

$$AFD_{DP} = \frac{\sum_{a \in \{DPclasses\}} g(a)}{\sum_{a \in \{DPclasses\}} f(a)} ,$$

$$AFD_{DPCluster} = \frac{\sum_{a \in \{DPclasses\} \cup \{DPrelations\}} g(a)}{\sum_{a \in \{DPclasses\} \cup \{DPrelations\}} f(a)} ,$$

$$AFD_{nonDP} = \frac{\sum_{a \in \{non-DPclasses\}} g(a)}{\sum_{a \in \{non-DPclasses\}} f(a)} ,$$

where $g(a)$ gives the total gaze time (total time of all fixations) in an area of interest and $f(a)$ gives the fixation count of an area of interest.

The task is typically solved in two phases. In the first phase, the subject tries to find the classes that participate in the design pattern. This is done through exploring all possible classes to find the relevant ones. In the second phase, they try to determine the roles of classes they suspect to be part of the design pattern. The first is an exploration phase where as the second is more focused.

8.1.5 Participants

The study participants were fifteen volunteers from the Department of Computer Science at Kent State University. There were seven undergraduates in their second year of study, six graduate students, and two faculty members. The undergraduates were considered to be novices in design whereas the graduates and faculty were experts and had more experience in the usage of UML as well as design patterns. Two of the subjects were female. All subjects had normal vision. Some wore contact or corrective lenses. The subjects were not aware of the experiment's hypotheses.

8.1.6 Instrumentation

The study was conducted in a dedicated room for the eye-tracking equipment. The subjects were seated approximately 60 cm away from the screen. An informed consent form was read and signed. The next step was calibrating the eye tracker for the subject. A five-point calibration was used (taking approximately one minute). During calibration, a subject focused their eyes on five points that appear on the screen (four for each corner, one for the center). The background color of the calibration was set to white since this was the background of the stimuli used in the study.

The first screen displayed instructions on what the task was. The next four screens, gave a description of each design pattern on the screen. They were allowed to study this for as long as they liked. After the subject understood the goal of the exercise, the actual study began. The moderator controlled the movement through the tasks to avoid any unnecessary timing delays between subjects. The subjects were asked to verbally state the role of each class in the design pattern. The experiment took eight

minutes on average with a maximum of thirteen minutes. Finally, after all the tasks were completed, the moderator collected some post-study data in an interview session.

8.2 Experimental Results

This section discusses the results from the eye-tracking study. The accuracy and time analysis is presented first, followed by the analysis of eye gaze data.

8.2.1 Accuracy and Time

The results of the experiment for accuracy and time are shown in Table 8.2. The Wilcoxon test was used since this is a within-subjects study. In this study, there is a significant difference with the multi-cluster layout for the Strategy and Singleton pattern with respect to the accuracy achieved. The previous study found support for the Strategy pattern but not the Singleton pattern in terms of accuracy.

It is surprising that the Singleton pattern benefits from the multi-cluster layout especially since it consists of only one class. There was one subtle difference in the Singleton pattern diagrams used in this study compared to the original study. A self-dependency relationship was added in a non-singleton class in both the multi-cluster and orthogonal layouts. This self-loop was included in the QObject class and the DrawApplication class in the multi-cluster and orthogonal layouts respectively. Even though these classes have a self-dependency, they are not part of the Singleton pattern. The non-singleton class with the self dependency had its attributes and methods visible in the orthogonal layout but were hidden in the multi-cluster layout. Even though these attributes and methods did not indicate that the class was a singleton, novices tended to

choose this class as their answer. We conjecture the Singleton pattern benefitted from the multi-cluster layout due to lower level of detail shown for this non-singleton class which caused subjects especially novices not to choose it as their answer and look elsewhere for another class. Novices tended to choose these classes erroneously since they tended to look for self-loops instead of looking at attributes and methods for a static instance of the class. The difference in the level of detail between the two layouts was unintentional and was only noticed after the study was conducted. Hence, results on the Singleton pattern need to be considered with caution.

The Observer pattern approaches significance for accuracy ($p\text{-value} = 0.059$) and is significant for relevance ($p\text{-value} = 0.045$). This shows us that the roles of Subject and Observer were mismatched in this study. In particular, the Observer role in the Observer pattern is detected significantly better in the multi-cluster layout ($p\text{-value} = 0.031$). In the post-interview session at the end of the study, subjects stated that Observer was difficult for them to detect. This remark is reflected in the data.

With respect to time, the multi-cluster layout is significantly better than the orthogonal layout in all four design patterns. This concurs with the results from the previous study. The effect sizes are much larger in this experiment than in the previous one. See Figure 8.3 for descriptive statistics.

We can reject the null hypothesis $H_{0_{\text{Accuracy}}}$ for the Strategy and Singleton pattern. Also, we can accept the alternative hypothesis to $H_{0_{\text{Time}}}$ for all four patterns, since in all four cases, the multi-cluster layout performs better and takes less time compared to the orthogonal layout.

Table 8.2. Eye-tracking study: 1-tailed Wilcoxon *p-values* (alpha=0.05) for accuracy, time, and relevance for each design pattern's role detection. Directionality implies that the multi-cluster layout performs better. Cohen's d denotes the effect size: 0.2(small), 0.5 (medium), >=0.8 (large). * indicates significance.

Role Detection Accuracy				Total Accuracy (Cohen's d)	Relevance (Cohen's d)	Time (Cohen's d)
Composite Pattern						0.021 * (0.64)
Roles	<i>Composite</i>	<i>Component</i>	<i>Leaf</i>			
	0.250	0.250	0.750	0.188 (0.2)	0.875 (0.06)	
Observer Pattern						0.019 * (0.73)
Roles	<i>Subject</i>	<i>Observer</i>				
	0.227	0.031 *		0.059 (0.6)	0.045 * (0.7)	
Strategy Pattern						0.036 * (0.70)
Roles	<i>Concrete</i>	<i>Context</i>	<i>Strategy</i>			
	<i>Strategy</i>					
	0.363	0.016 *	0.016 *	0.045 * (0.8)	0.424 (0.1)	
Singleton Pattern						0.024 * (0.40)
Role	<i>Singleton</i>					
	0.031 *			0.031 * (0.9)	0.031 * (0.9)

Figure 8.3. Accuracy and time box plots for patterns across layouts for the eye-tracking study

8.2.2 Visual Effort

We measure visual effort using eight variables, each of which use two main types of eye gaze data: the fixation count on each diagram and the average fixation duration.

Table 8.3 shows the results of the Wilcoxon test for each measure. We consider each of these measures separately.

Table 8.3. 1-tailed Wilcoxon p -values ($\alpha=0.05$) for the visual effort measures. Directionality implies the multi-cluster layout performs better.

Area of Interest	Dependent Variable	Composite	Observer	Strategy	Singleton
Entire Diagram	FC_T	0.014 *	0.042 *	0.126	0.738
DP Classes	FR_{DP}	0.076	0.036 *	0.165	0.994
Non-DP Classes	FR_{nonDP}	0.932	0.968	0.849	0.008 *
DP Cluster	FR_{DPCluster}	0.281	0.032 *	0.042 *	0.996
Entire Diagram	AFD_T	0.533	0.015 *	0.021 *	0.195
DP Classes	AFD_{DP}	0.165	0.211	0.042 *	0.700
Non-DP Classes	AFD_{nonDP}	0.002 *	0.001 *	0.874	0.018 *
DP Cluster	AFD_{DPCluster}	0.281	0.047 *	0.024 *	0.423

Entire Stimulus: With respect to the area of interest which is composed of the entire diagram (denoted by the subscript T), there is a significant higher fixation count (FC_T) in the orthogonal layouts for the Composite and Observer patterns. See Figure 8.4 for the distribution of the first and fifth rows in Table 8.3. The Strategy and Singleton do not display a significant difference in fixation counts in the entire diagram. In terms of duration (AFD_T), in the Observer pattern and the Strategy pattern, subjects spend much more time in the orthogonal layout overall than the multi-cluster layout. No statistical difference was found in the Composite and Singleton pattern, however we did note more time spent in the orthogonal layouts. The measures in this area of interest give an overall indication of visual effort.

Figure 8.4. Box plots for FC_T and AFD_T

Design Pattern Classes and Design Pattern Clusters: The measures in the DP and DPcluster areas of interest focus on phase 2 of the task where the subject is trying to identify roles after they have identified classes that participate in the design pattern. Boxplots for $FR_{DPcluster}$ and $AFD_{DPcluster}$ are shown in Figure 8.5. The fixation rate in design pattern clusters is significantly lower using the multi-cluster layout for the Observer and Strategy patterns (p-values= 0.032 and 0.042 respectively). The average

fixation duration in design pattern clusters is also significantly lower using the multi-cluster layout for the Observer and Strategy patterns (p-values= 0.047 and 0.024 respectively). The Composite and Singleton patterns are not significant. The higher the fixation rate and average fixation duration in the cluster, the more difficult it is to determine the roles.

Considering the design pattern classes only (DP), we find the Observer pattern had a lower fixation rate for the multi-cluster layout and the Strategy pattern had a lower average fixation duration for the multi-cluster layout. No other differences were reported.

Since the design pattern cluster covers a larger area than the design pattern classes, more time is spent looking at the cluster than on the design pattern classes themselves. The DP area of interest excludes the fixations on relationship ends. The measures $FR_{DPCluster}$ and $AFD_{DPCluster}$ give a more accurate picture than the FR_{DP} and AFD_{DP} , since they focus on the classes and relationships. Even though fixations are usually not found on the relationship lines, they are most often found at the relationship ends. These fixations fall outside the bounding box of the class and are not counted in the DP area of interest but are counted in the DPCluster AOI.

Figure 8.5. Box plots for $FR_{DPCluster}$ and $AFD_{DPCluster}$

Non-DP Classes: The measures in this area of interest focus on phase one of the task, where the subject is looking for the classes that belong to a particular design pattern. The higher the rate in non-design pattern classes, the more difficult it is to spot the classes in the pattern, i.e., they explored more classes before selecting their answer. The fixation rate for non-design pattern classes, FR_{nonDP} , shows a significantly higher effort in the orthogonal layout for the Singleton pattern ($p-value = 0.008$). This is also reflected in the accuracy result above. The average fixation duration for non-design pattern classes, AFD_{nonDP} , is significantly higher in the orthogonal layout for the Composite, Observer, and Singleton patterns ($p-values=0.002, 0.001, \text{ and } 0.018$ respectively). This implies that trying to search for the relevant classes took much longer in the orthogonal layout than the multi-cluster layout for these patterns.

Based on the results, we can reject the null hypothesis $H0_{VisualEffort}$ for the Observer and Strategy patterns, where the multi-cluster layout is shown to be reduce visual effort.

The average number of fixations for each diagram layout is shown in Figure 8.6 along with the mean of the average fixation duration. In the case of Composite, Observer and Strategy, a much higher rate is seen (double in case of the Observer pattern). The average fixation duration shows a higher amount of time spent in the Observer, Strategy and Singleton patterns.

a)

b)

Figure 8.6. The average number of fixations and the average fixation duration for each layout across the four design patterns

8.2.3 Qualitative Analysis

One of the research questions (RQ5) posed in Section 1.2 seeks to determine the effect layout of a class diagram has on eye gaze behavior of experts and novices. This section addresses this question via a qualitative analysis. A quantitative analysis is given in the previous section.

In this study, the eye gaze data is mainly tracked at the class level. The eye tracker gives information at the method level of granularity as well. Heat maps and gaze plots are analyzed to get a qualitative view of the eye gaze data. A heat map is a technique to visualize gaze behavior of a group of subjects. A heat map is superimposed on top of the diagram and highlights areas where subjects have been looking. Red indicates the highest percentage and green indicates the lowest. A heat map can be based on the fixation count or fixation length. Fixation length is used in this section. These maps are best viewed in color. A gaze plot displays a static view of the eye gaze data for each diagram. It is useful to visualize scan paths. A scan path is a directed sequence of fixations. A fixation is illustrated using a circle where the radius represents the length of the fixation.

A cumulative heat map of all eight experts participating in this study is shown in Figure 8.7 for the Strategy pattern in both layouts. The multi-cluster layout on the left is from *Qt* and the paired diagram on the right in orthogonal layout is from *JHotDraw*. The highlighted areas correspond to the relative fixation length. These maps clearly show the difference in time spent by experts in the two different layouts. Eye fixations are mainly seen on the classes participating in the design pattern for the multi-cluster layout, whereas in the orthogonal layout, almost half the number of classes in the diagram are looked at for a considerably more time compared to the multi-cluster layout. The visual effort in fixation length is clearly higher for the orthogonal layout.

a) multi-cluster layout

b) orthogonal layout

Figure 8.8. A heat map based on fixation length of all seven novices for the Strategy pattern

In order to compare experts with novices, a cumulative heat map of all seven novices participating in this study is shown in Figure 8.8 for the Strategy pattern in both layouts. These correspond to the images shown in Figure 8.7. The multi-cluster layout on the left is from *Qt* and the paired diagram on the right in orthogonal layout is from *JHotDraw*. A difference is noticed in the multi-cluster layout between experts and novices. Novices tend to choose an aggregation relationship related to the parent of a generalization hierarchy by purely matching the template of the Strategy pattern. The five classes fixated on in the multi-cluster layout are not part of the Strategy pattern. Novices do not look at method names or member variables, but only on the class name and relationships. Within the novice category, there is a higher effort seen in the orthogonal layout when compared to the multi-cluster layout. It is important to note that the classes mostly focused on the orthogonal layout are also not part of the Strategy pattern. This tends to indicate that experience interacts with layout to have an effect on

the performance, however this analysis is not done statistically due to ANOVA assumption violations.

a) multi-cluster layout b) orthogonal layout

Figure 8.9. A heat map based on fixation length of all fifteen subjects for the Observer pattern

A cumulative heat map of all fifteen subjects participating in this study is shown in Figure 8.9 for the Observer pattern in both layouts. The multi-cluster layout on the left is from *JHotDraw* and the paired diagram on the right in orthogonal layout is from *JUnit*. The highlighted areas correspond to the relative fixation length. These maps clearly show the difference in time spent by all subjects in the two different layouts. Eye fixations are mainly seen on the classes participating in the Subject and Observer roles in the design pattern for the multi-cluster layout, whereas in the orthogonal layout, almost half the number of classes in the diagram are looked at for a considerably more time compared to the multi-cluster layout. The visual effort in fixation length is again clearly higher for the orthogonal layout.

Figure 8.10. A gaze plot of an expert for the Observer pattern

A gaze plot of an expert is shown in Figure 8.10. The duration of each fixation (a blue circle on the diagram) is represented by the radius. This plot clearly shows a high number of fixations for the orthogonal layout compared to the multi-cluster layout. Since the Subject and Observer are placed further apart in the orthogonal layout, this causes undue switching between two opposite sides of the diagram as can be seen in the orthogonal layout. The visual effort is much higher for the orthogonal layout when compared to the multi-cluster layout.

Figure 8.11. A gaze plot of a novice for the Observer pattern

A gaze plot for a novice is shown in Figure 8.11 for the same two diagrams as Figure 8.10 in order to compare eye gaze behavior of experts and novices. The duration of each fixation is much smaller for experts than novices, even though experts might have a higher number of total fixations. The diagrams correspond to the ones given in Figure 8.10, the only difference being the type of data superimposed on top. After analyzing the plots, it seems that the multi-cluster layout has a fewer number of fixations than the orthogonal layout. Also, the length of the fixations is much larger for the orthogonal layout. The visual effort is again higher for the orthogonal layout when compared to the multi-cluster layout.

a) multi-cluster layout

b) orthogonal layout

Figure 8.12. A gaze plot of an expert for the Strategy pattern

This last pair of gaze plots in Figure 8.12 show an expert's eye gaze data for the Strategy pattern. Fixations are found on the three classes that participate in the design pattern for the multi-cluster layout. The orthogonal layout on the other hand, has a much higher number and higher duration of fixations. Every class is analyzed. Visual effort is again clearly higher for the orthogonal layout compared to the multi-cluster layout.

One example of where experts differ from novices is in the identification of the Singleton pattern. See Figure 8.13 for an example of a gaze plot. It shows an expert looking at the attributes and methods to determine the answer. The novice on the other hand mainly focuses on the class name.

Figure 8.13. Portion of a stimulus comparing an expert and novice gaze plot for Singleton pattern.

In summary, classes participating in a design pattern in the multi-cluster layout act like beacons drawing the attention of subjects thereby allowing them to complete the task in less time.

8.3 Discussion

The multi-cluster layout has a positive effect on accuracy, speed and visual effort needed to solve the role detection tasks. In particular, the Strategy and Observer patterns benefitted the most. We could not reject the null hypotheses for the Composite and Singleton patterns.

The Strategy pattern consists of an aggregation and generalization relationship. The aggregation is not within the hierarchy like the Composite pattern. Since the

aggregation is not within the same hierarchy in the Strategy pattern, the class participating in the *Context* role tends to be placed further away in the orthogonal layout. The multi-cluster layout on the other hand, recognizes cohesive clusters and places classes with aggregations closer together even though this might violate an aesthetic criteria of placing all children at the same level under the parent. This is the main factor that causes the Strategy pattern to have a higher accuracy with less time spent in role detection for the multi-cluster layout. Since the Composite pattern has its aggregation within the hierarchy, it tends to be positioned similarly in both layouts. The same reasoning applies to the Observer pattern, where the orthogonal layout tends to place the Subject and Observer connected by an association further apart, requiring more effort in tracing the relationship.

Considering all the visual effort measures, we find more support for the Strategy and Observer patterns using the multi-cluster layout. Results also indicate more time spent looking for classes (phase one: AFD_{nonDP}) involved in the design pattern in the orthogonal layout. Based on the eye gaze data, we find that novices and experts had different techniques to identify patterns and roles. Novices used a template matching method and tried to match the template to the diagrams occasionally looking at attributes and methods. Experts focused on method names and attributes in addition to the class names. This is more evident in the Singleton and Observer patterns. In order to identify Singletons, two criteria need to be met a) a self-association, and b) static instance. The novices only looked for self associations and this by itself does not indicate a Singleton

class. Experts looked for the static instance first, which is the main reason behind the self association.

It is important to use both fixation counts and gaze duration to determine the effort since based on our results we find that even though the fixation counts may not be significantly different, the duration is significant and vice versa.

The post questionnaire collected information about the difficulty level of the systems used and the role detection in design patterns. None of the subjects (except one) were familiar with the design of the systems. None were aware of design pattern usage in the systems. The Singleton pattern was considered to be the easiest, followed by the Composite pattern. The Strategy and Observer patterns were ranked at a higher level of difficulty.

8.4 Threats to Validity

This section discusses threats to validity and measures taken to minimize them.

Internal Validity: To minimize learning effects, a different subject system was used for each layout due to the within-subjects nature of this study. The study was designed to be completed in less than 15 minutes to avoid any fatigue effects. Each role detection question did not contain any other patterns i.e., Strategy detection diagrams did not contain a Composite pattern to avoid any confounding factors. Since this is an eye-tracking study, the reading method used by subjects might affect the results of detecting the roles of design patterns. A top-down scan versus a left-right scan of the diagram might affect the results. However, we did notice that subjects looked at almost all classes in the diagram before making their choice. The manually engineered diagrams might

pose a threat, hence the number of crossings were kept the same across layouts. The number of classes and relationships were also the same across layouts. Another threat to validity is the possible overlap in areas of interest. Sometimes, more than one object was part of the area of interest, making it difficult to know which object was actually being looked at. This was due to the very nature of the UML class diagram layout. Care was taken to minimize overlaps where possible.

External Validity: We used students and faculty as our sample population. They were all familiar with design patterns at varied levels of expertise. This allowed us to compare novices with experts. Many of the subjects worked in industry and are comparable to senior developers. Another concern with regards to external validity is with respect to representative tasks. The tasks were based on real open-source systems (not toy applications) and hence more representative of design pattern usage in software.

Construct Validity: Since visual attention is related to mental processing of the information [Just, Carpenter 1980], the measures derived from the fixation counts and durations should be valid. Eight measures were used to avoid mono-method bias. The visual effort measures for duration, use an average since the areas of interest are fairly large compared to eye tracking studies done in psychology where the area of interest is a word. Another option would be to use the sum to increase the power of the statistical tests. However, we do not believe that using the sum of durations would decrease the statistical power.

Conclusion Validity: Due to low sample size, we use the paired Wilcoxon test for hypotheses testing.

8.5 Summary

An eye-tracking study is conducted to determine the effect layout has on the detection of roles in design patterns. This was a replication of the questionnaire-based study conducted in the previous chapter interchanging the type of layout for each pattern. Visual effort is determined via a set of eight measures and provide an objective metric to measure the quality of UML class diagram layouts. Both studies report higher accuracy in the case of the Strategy pattern for the multi cluster layout. In addition, this study also reports a higher accuracy for the Observer role in the Observer pattern. In particular, the Observer pattern roles seem to be mismatched. This could be related to the fact that the layouts used were flipped in this study or due to the type of sample studied. All four patterns report lower time spent on task in the multi-cluster layout. Results also indicate a lower visual effort in design pattern clusters for the Strategy and Observer patterns in the multi-cluster layout. In future work, we plan to expand the study to include other design patterns in other application domains. We also plan on investigating the relationship of class stereotypes in the design pattern role detection with respect to maintenance tasks. Investigating the level of detail with respect to layout is another area of future work.

CHAPTER 9

An Eye-tracking Study on the Effect of Identifier Styles on Comprehension

An empirical study to determine if identifier-naming conventions (i.e., CamelCase and under_score) affect code comprehension is presented. An eye tracker is used to capture quantitative data from human subjects during an experiment. The intent of this study is to replicate a previous study published at ICPC 2009 (Binkley et al.) that used a timed response test method to acquire data. The use of eye-tracking equipment gives additional insight and overcomes some limitations of traditional data gathering techniques. Similarities and differences between the two studies are discussed. One main difference is that subjects were trained mainly in the underscore style and were all programmers. While results indicate no difference in accuracy between the two styles, subjects recognize identifiers in the underscore style more quickly.

9.1 Motivation

UML class diagrams mainly consist of classes and relationships. Each class presents data members and operations. See Figure 1.1. The data members are identifiers used in the code to encapsulate behavior and functionality. Identifiers make up more than 1/3rd of the space of each class (since they are used in method calls as well) and are used to understand relationships such as generalizations, associations, and dependencies. If a certain identifier style has an effect on readability, this also impacts the readability of class diagrams, since there are many identifiers present in both the member variables

compartment and the operations compartment of a class in the form of parameters. This study seeks to determine if identifier style could possibly affect the reading of UML class diagrams. It does not consider layout or class diagrams in the materials tested. A simple reading exercise is done instead. However since the same type of reading occurs in class diagrams, these results may be extended to the readability of both source code and class diagrams.

9.2 Overview

The comprehension of identifier names in programs and UML class diagrams is at the core of program understanding. Identifier names are often key beacons to program plans that support higher-level mental models of understanding. According to Deußenböck et al. [Deußenböck, Pizka 2006] identifiers make up approximately 70% of source code. If a certain identifier naming style significantly increases the speed of code comprehension, this could significantly impact overall program understanding.

Currently we have two main styles for identifiers, namely camel-case (e.g., `studentGrade`) and underscore (e.g., `student_grade`). In the work presented here, we study the comprehensibility of these two styles and attempt to determine if one is significantly better than the other. Our goal is to add to the basic understanding of how we comprehend identifiers so that coding standards [Sutter, Alexandrescu 2004] can reflect the most efficient techniques.

Early programming languages such as Basic, COBOL, Fortran, Pascal, and Ada were case insensitive and programmers were encouraged to use underscores to separate compound identifier names. With the advent of case-sensitive languages such as C, C++,

Python, and Java, the trend has been to use the camel case style identifiers. This may in part be due to the fact that it is a bit faster to type a camel-case identifier than it is an underscore identifier. This is due to the position of the underscore on the keyboard and the number and combination of keystrokes required. However, does the ease of writing identifiers affect the accuracy of code readability and maintainability?

To address this topic, Binkley et al. [Binkley, Davis, Lawrie, Morrell 2009] conducted a study with 135 subjects consisting of programmers and non-programmers to determine which identifier style was faster and more accurate. They hypothesized that identifier style affects the speed and accuracy of software maintenance. The subjects (who had programming experience) were mostly trained in the camel-case style. The study used an online game-like interface to gather timed responses from the subjects. Their findings show that camel casing leads to higher accuracy among all subjects, and those trained in camel case, were able to recognize camel-cased identifiers faster. However, with respect to all subjects, camel-cased identifiers took 13.5% longer than underscored identifiers ($p\text{-value} < 0.0001$).

Here, we attempt to replicate Binkley et al.'s [Binkley, Davis, Lawrie, Morrell 2009] experiment using an eye tracker to gather eye gaze data during the experiment. In our study, only programmers (experts and novices) are used as subjects. All of our subjects had experience with both styles and their preferences of style was approximately split even among the group. The main task of the study remains the same as Binkley's, which is to pick the correct identifier from a group of four closely related, although different, identifier names. Results from eye tracking studies done in the domain of

cognitive psychology [Epelboim et al. 1997; Rayner, Fischer, Pollatsek 1998] on reading un-spaced text imply that camel casing should be more difficult to read (than underscore). Replicating the experiment using an eye tracker will add to the empirical evidence as to which style is faster and more accurate.

9.3 Research Questions

The goal of this study is to analyze human subjects' eye-gaze data while they perform the tasks of correctly detecting an identifier from a group of four closely related identifiers. Although this task is relatively simple as pointed out by Binkley et al. [Binkley, Davis, Lawrie, Morrell 2009], it gives insight into the readability aspect of identifier styles. With the data generated from an eye tracker we know the exact location of where the subject is looking, the duration of the subject's gaze at a particular location, and movement between different locations on the screen. These measures lead to a fine-grained analysis thus generating more refined conclusions. Although there are many eye tracking studies related to evaluating user interfaces, [de Kock, van Biljon, Pretorius 2009; Goldberg et al. 2002; Matsuda, Uwano, Ohira, Matsumoto 2009], there are very few studies on how programmers read and comprehend source code [Bednarik, Tukiainen 2006; 2008; Crosby, Stelovsky 1990; Uwano, Nakamura, Monden, Matsumoto 2006]. To bridge this gap, we conducted an eye-tracking replication of Binkley et al.'s [Binkley, Davis, Lawrie, Morrell 2009] study since the topic lends itself well to eye tracking analysis.

The main research questions this study addresses are:

- RQ9.1: Does identifier style affect the accuracy and time needed to read and detect correct identifiers?
- RQ9.2: Is the visual effort needed to read and detect correct identifiers the same for the camel case and underscore styles?

9.4 Experimental Design

The experiment seeks to analyze the effect identifier style has on searching for correct identifiers for the purpose of evaluating their usefulness in code readability and comprehension with respect to effectiveness (accuracy) and efficiency (time) from the point of view of the researcher in the context of students at Kent State University.

Table 9.1. Experiment overview for studying identifier styles

Goal	Study the effect identifier style has on code readability
Main Factor / Independent variable	Identifier style (Style) with two treatments: camel case or under score
Dependent variables	Correctness, Find Time, Visual Effort
Secondary factors	Reading Time, Experience, Phrase Length, Phrase Origin, Style Preference, Visual Effort on Reading Phrase

An overview of the experiment is given in Table 9.1. The main factor being analyzed is the identifier style used. The study variables are discussed in Section 9.4.3. While analyzing the results we also examined secondary factors such as the subject's experience. The experiment is conducted as a within-subjects crossover design where all subjects are given all types of treatments of the factor and a paired comparison between identifier styles is made for each subject. In Binkley's experiment, repeated measures were used however the exact details of the number of subjects in each group are not

given. Since our sample size is relatively small ($N=15$) compared to the Binkley's, we wanted to gather more data points for each style and use a within-subjects comparison.

9.4.1 Eye-tracking Apparatus

Refer to Section 8.1.1 on page 175 for details on the eye tracker used in this study.

9.4.2 Material and Stimuli

The main objects of this study are a set of eight phrases (same as Binkley's study). The subject first reads a phrase and when they are done studying it, the next screen asks them to choose an identifier (from four choices) that exactly matches the phrase they just saw. Figure 9.1 shows the *phrase stimulus* and the *question stimulus* for each task. There are eight such tasks. See Table 9.2 for the set of phrases used. Only one of the choices is correct, the rest are distracters that change the beginning, middle and end of the identifier. For detailed information about the identifier selection process and distracters used, we direct the reader to [Binkley, Davis, Lawrie, Morrell 2009]. Unlike the Binkley's study, the clouds on the question stimuli do not move and the phrase is not shown on the question stimuli. Since the previous study does not indicate which style was used to generate identifiers for each phrase, we randomly assigned a style to each phrase within each phrase type.

Each of the identifier phrases is characterized by a length, origin, and style used. The length is the number of words in the phrase. Phrase origin determines whether or not the phrase is likely to be in source code. For example, *river bank* is a 2-word non-code

phrase since the probability of finding it in source code is low. The reason for including non-code phrases in the original study was to determine if familiarity with a phrase had an effect on performance.

Table 9.2. Phrases used in the study.

Phrase type	ID	Style	Phrase	Distracters Used (Begin, Middle, End)
2-word code	Q1	CamelCase	start time	smart time, start mime, start tom
	Q5	under_score	full pathname	fill pathname, full mathname, full pathnum
2-word non-code	Q7	CamelCase	river bank	riser bank, river tank, river ban
	Q3	under_score	drive fast	drove fast, drive last, drive fat
3-word code	Q4	CamelCase	extend alias table	expand alias table, extend alist table, extend alias title
	Q2	under_score	get next path	got next path, get near path, get next push
3-word non-code	Q6	CamelCase	movie theater ticket	mouse theater ticket, movie thunder ticket, movie theater ticker
	Q8	under_score	read bedtime story	raid bedtime story, read bedstore story, read bedtime store

The presentation order of the questions is shown in the second column of Table 9.2. The order was determined using Latin squares to avoid learning biases. During the analysis, we do a pair-wise comparison between the four pairs: Q1 and Q5, Q7 and Q3, Q4 and Q2, Q6 and Q8. Instead of testing each subject on the camel case and underscore versions of the same identifier (causing learning effect), a different but similar identifier in the opposing style is used.

Figure 9.1. Phrase Stimulus (a) with task description in bold. Question Stimulus (b) with clouds to detect the correct identifier formed using the phrase

9.4.3 Variables

The study consists of one independent variable, *identifier style*. The two values associated with this main factor are camel case and underscore. The dependent or response variables are described next.

- Correctness: Denotes the accuracy of the answer verbally stated by a subject.

- Find Time (FT₀): Denotes the time taken by a subject to verbally state the correct answer. This is recorded in milliseconds.

The idea behind eye tracking is that visual attention (focus on a particular location) triggers mental processes to comprehend or solve a given task [Just, Carpenter 1980]. Based on this correlation, we can study the cognitive behavior and effort involved in solving a task. For each *phrase stimulus*, two areas of interest (AOI) are created: one for the task description on top and one for the phrase to be studied. For each *question stimulus*, five areas of interest are created: one for each cloud and one for the task description on top. The areas of interest are represented as rectangles enclosing the task description, phrase and clouds and were constructed with a buffer zone of at least 50 pixels to accommodate for any small drifts of the eye tracker.

Visual Effort: Denotes the amount of effort needed to verbally state the correct answer. We gather the following data for each area of interest on each stimuli as well as a global measure for the stimuli on the whole, which is the sum of all the areas of interest. We analyze our results only based on areas of interest and not on eye gaze data on the blank part of the screen. Two main types of eye gaze data are eye fixations and saccades. A *fixation* is the stabilization of the eye on an object of interest for a period of time, whereas *saccades* are quick jerky movements from one fixation to another. It has been determined that comprehension mainly takes place during fixations and not during saccades. The eye tracker was set to filter fixations within 20 pixels with duration of at least 40ms. This is the standard setting recommended for reading. Visual effort is determined using each of these individual measures.

- Fixation Count of Question Stimulus ($\mathbf{FC}_Q$): The total number of eye fixations on all five areas of interest on the *question stimulus*.

$$\mathbf{FC}_Q = \sum_{a \in \{task, allcloud\}} f(a) ,$$

where $f(a)$ gives the fixation count of an area of interest.

- Fixation Rate of Correct Identifier ($\mathbf{FR}_{correct}$): The total number of eye fixations on the correct identifier cloud with respect to all four clouds on the *question stimulus*.
- Fixation Rate of Distracters ($\mathbf{FR}_{distracters}$): The total number of eye fixations on the distracter clouds with respect to all four clouds on the *question stimulus*.

A higher fixation count and fixation rate indicates more effort needed by subjects to solve the task. These measures are characterized as follows.

$$\mathbf{FR}_{correct} = \frac{\sum_{a \in \{correctcloud\}} f(a)}{\sum_{a \in \{correctcloud\} \cup \{distractercloud\}} f(a)}$$

$$\mathbf{FR}_{distracter} = \frac{\sum_{a \in \{distractercloud\}} f(a)}{\sum_{a \in \{correctcloud\} \cup \{distractercloud\}} f(a)} ,$$

where $f(a)$ gives the fixation count of an area of interest.

- Average Fixation Duration on Question Stimulus: ($\mathbf{AFD}_Q$): The average length of time of all fixations in all five areas of interest on the *question stimulus*.
- Average Fixation Duration on Correct Identifier: ($\mathbf{AFD}_{correct}$): The average length of time of all fixations on the correct identifier cloud on the *question stimulus*.

- Average Fixation Duration on Distracters: ($\mathbf{AFD}_{\text{distracters}}$): The average length of time of all fixations on the distracter clouds on the *question stimulus*.

The unit of measure is milliseconds. The more time spent analyzing the stimuli in search of an answer indicates more effort needed by subjects to solve the task. Each of the above measures is shown below.

$$\mathbf{AFD}_Q = \frac{\sum_{a \in \{\text{task}, \text{allcloud}\}} g(a)}{\sum_{a \in \{\text{task}, \text{allcloud}\}} f(a)}, \quad \mathbf{AFD}_{\text{correct}} = \frac{\sum_{a \in \{\text{correctcloud}\}} g(a)}{\sum_{a \in \{\text{correctcloud}\}} f(a)}$$

$$\mathbf{AFD}_{\text{distracter}} = \frac{\sum_{a \in \{\text{distractercloud}\}} g(a)}{\sum_{a \in \{\text{distractercloud}\}} f(a)},$$

where $g(a)$ gives the total gaze time (total time of all fixations) in an area of interest and $f(a)$ gives the fixation count of an area of interest.

Secondary variables are factors that might interact with the independent variable to have an effect on the dependent variables. These are described next.

- Phrase Length: The length of the phrase is determined by the number of words in the phrase. Phrases of length two and three are used.
- Phrase Origin: Determines whether or not the phrase is likely to be in source code. Possible values include *code* and *non-code*.
- Reading Time ($\mathbf{RT_P}$): Denotes the time (ms) taken by a subject to study a phrase on the *phrase stimulus* before proceeding to the *question stimulus*.

- Experience: Indicates the level of expertise of the subjects. Two levels experts and novices, were determined based on programming experience, number of years worked, and number of years in Computer Science. This variable combines the *Years Worked* variable and the *Training* variable used in the Binkley's study. The difference here is that we look for expertise within programmers rather than looking for differences between programmers and non-programmers.
- Style Preference: Denotes a subject's identifier style preference. Three values associated with this variable are camel case, underscore, and no preference.

Visual Effort on Reading Phrase: Denotes the amount of effort needed to study a phrase on the *phrase stimulus*. It is measured using each of the following variables.

- Fixation Count of the Phrase AOI (FC_P): The total number of eye fixations on the phrase area of interest on the *phrase stimulus*. This does not include fixations on the task. It is denoted by $FC_P = \sum_{a \in \{phrase\}} f(a)$, where $f(a)$ gives the fixation count of an area of interest.

- Average Fixation Duration on the Phrase: (AFD_P): The average length of time of all fixations in the phrase AOI on the *phrase stimulus*. It is denoted by

$$AFD_P = \frac{\sum_{a \in \{phrase\}} g(a)}{\sum_{a \in \{phrase\}} f(a)}, \text{ where } g(a) \text{ gives the total gaze time in an area of}$$

interest and $f(a)$ gives the fixation count of an area of interest.

In this study, we did not measure the amount of time spent on demographics (conducted at the end of the study) and the age of subjects. The questions on

demographics were conducted verbally and in an interview-like setting. This time varied greatly depending on the verbiage used. Since this was more open ended, we do not include it as a variable in our analysis.

9.4.4 Hypotheses

Based on the research questions posed in Section 9.3, we generate the following null hypotheses. See Table 9.3. The alternative hypotheses do not assume directionality and simply state that the distribution is not same between identifier styles.

The first two hypotheses are the same as presented in Binkley's study. $H1_0$ seeks to determine if identifier style has an effect on correctness. In this case, correctness refers to the subject accurately stating the correct identifier built using the corresponding phrase.

Table 9.3. Null hypotheses for the identifier style study

H1₀	Correctness is the same regardless of the Style of the identifier.
H2₀	Find Time is the same regardless of the Style of the identifier
H3₀	The effect of Style on Correctness is independent of Experience
H4₀	The effect of Style on Find Time is independent of Experience
H5₀	Visual Effort is the same regardless of the Style of the identifier
H6₀	The effect of Style on Visual Effort is independent of Experience

The second hypothesis ($H2_0$) seeks to determine if identifier style has an effect on the Find Time. In this case, Find Time refers to the time needed to verbally choose an answer from the *question stimulus*.

Hypotheses 3 (H3₀) and 4 (H4₀) are similar to the previous study using the *Experience* variable instead of *Training*. They seek to determine if experience interacts with identifier style to have an effect on Correctness and Find Time.

The last two hypotheses relate to eye-tracking measures defined in the previous section. Hypothesis 5 (H5₀) seeks to determine if identifier style has an effect on the visual effort necessary to solve the task of recognizing the correct identifier. Finally, hypothesis 6 (H5₀) investigates the interaction of the secondary variable Experience on solving the task.

9.4.5 Participants

The study participants were fifteen volunteers from the Department of Computer Science at Kent State University. There were seven undergraduates in their second year of study, eight graduate students and two faculty members. Two of the subjects were female. Subjects were historically trained mostly in the underscore identifier style and were all programmers. All subjects had normal vision. Some wore contact or corrective lenses. The subjects were not aware of the experiment's hypotheses.

The following demographic data was collected for each subject after the study was completed: years in the CS program, years of experience in programming, years of working experience and identifier style preference. Based on this information, two groups of expert and novice programmers are determined.

9.4.6 Instrumentation

The study was conducted in a dedicated room for the eye-tracking equipment. The subjects were seated approximately 60 cm away from the screen. An informed consent form was read and signed. The next step was calibrating the eye tracker for the subject. A five-point calibration was used (taking approximately one minute). During calibration, a subject focused their eyes on five points that appear on the screen (four for each corner, 1 for the center). The background color of the calibration was set to white since this was the background of the stimuli used in the study.

The first screen displayed instructions on what the task was. Next, two sample questions: one camel case and one underscore, illustrating how to answer the questions were presented. After the subject understood the goal of the exercise, the actual study began.

For each of the eight tasks, the *phrase stimulus* was presented first followed by the *question stimulus* (See Figure 9.1). After the subjects were done studying the *phrase stimulus*, they said “next” to proceed to the *question stimulus*. The moderator controlled the movement through the tasks to avoid any unnecessary timing delays between subjects. The subjects were asked to verbally state the answer using the letter (i.e., A, B, C, or D) placed on top of the identifier in the cloud. After the eight identifier recognition tasks, an object location task was administered (See Section 9.5.3). The experiment took 13 minutes on average. Finally after all the tasks were completed, the moderator debriefed each participant to gather some demographic data in an interview-like manner. This concluded the experiment.

9.5 Experimental Results

This section presents the results of the experiment with respect to each dependent variable. In order to facilitate comparison to the Binkley study, a linear mixed-effects regression model is fit to the Find Time dependent variable. In addition, since our study used within-subjects, the non-parametric paired Wilcoxon test is used to determine significance for all dependent variables. Effect sizes using Cohen's d are noted to make results comparable with future studies on the topic.

9.5.1 Correctness and Find Time

Only one subject answered one question (Q4) incorrectly. This question used the phrase *extend alias table* in camel case style. The subject chose *extendAliasTitle* (distracter at the end of the identifier). This is in line with the distracter analysis done in [Binkley, Davis, Lawrie, Morrell 2009] that reports mistakes in camel casing occur more frequently when a change occurs at the end of the phrase. All other subjects answered the questions correctly. In this case, we cannot reject the null hypothesis ($H1_0$) and the subsequent related hypothesis ($H3_0$). This implies that there is no difference in accurately recognizing an identifier in either style. No further statistical analysis is needed in this case.

We now investigate the second hypothesis ($H2_0$) which examines the effect of identifier style on the speed of finding the correct identifier. Figure 9.2 presents the distribution of the Find Time dependent variable.

The data for Find Time was found to be normally distributed using the *Shapiro-Wilk* normality test. Similar to the original study, a simple linear mixed-model (at 95%

confidence) is first fit to the data, where only Style is considered as an explanatory variable. In the simple model, the parameter estimate for Style is statistically significant (Style p -value=0.015, Intercept p -value <0.0001). On average, camel-cased identifiers took 932 ms (20%) longer than underscored identifiers. In this case, we can reject the null hypothesis (H_{20}).

Figure 9.2. Descriptive statistics for Find Time in each category of identifier Style (CC=CamelCase and US=Under score)

A second model was fit to the data and included the secondary variable Experience as an explanatory variable in addition to Style, to determine if it interacts with Style to have an effect on Find Time (H_{40}). In this model, Style is still statistically significant (p -value=0.022). See Table 9.4. However, Experience does not significantly

interact with Style to have an effect on Find Time ($p\text{-value} = 0.394$). In this case, we cannot reject the null hypothesis (H_{40}).

Table 9.4. Model parameters for interaction between Experience and Style

Variable	Value	Standard Error	t	p-value
Intercept	4311.85	651.02	6.62	<0.0001
Style	1275.5	550.864	2.31	0.022
Experience	364.83	891.454	0.40	0.683
Style * Experience	-644.65	754.302	-0.85	0.394

Even though the result is not statistically significant, we can make some observations about the findings. The interaction plot is given in Figure 9.3. There is a larger time difference between experts and novices with respect to camel-cased identifiers, whereas the difference is less for underscored identifiers. Another observation is that the difference in time between identifier styles within experts is much less compared to the difference for the novice category. This implies that experts are not affected as much as novices by the identifier style used.

Figure 9.3. Interaction between subjects' experience and identifier style.

Finally, we fit a complex model to the Find Time response variable to mimic the analysis of the original study. To determine if there are any other confounding variables, this complex model includes all secondary variables common to the original study discussed in Section 9.4.3. This included Style, Style Preference, Experience, Phrase Origin, Read Time, and Phrase Length including all interactions between Style and each variable. The interaction between Phrase Origin and Experience is also included.

After backward elimination of non-significant terms, the model is presented in Table 9.5. This model confirms the significance of Style and the non-significance of Experience and Style*Experience. In addition, it shows that Reading Time is not significantly different for the phrases used to construct different identifiers. However, Phrase Length significantly interacts with Style to have an effect on Find Time. Identifiers consisting of phrases of length three take 45% longer than phrases of length two. We also see that phrases of length two for camel-cased identifiers take approximately the same time as corresponding underscore identifiers however, for phrases of length three camel-cased identifiers take 36% longer than corresponding underscore identifiers.

Table 9.5. Complex Model parameters for Find Time

Variable	Value	Standard Error	t	<i>p-value</i>
Intercept	4790.6	867.89	5.52	<0.0001
Style	2159.9	571.22	3.781	0.0002
Experience	364.9	869.74	0.42	0.67
Reading Time	-0.001	0.15	-0.007	0.99
Phrase Length	-950.9	459.7	-2.068	0.04
Style * Phrase Length	-1768.8	647.0	-2.734	0.007
Style * Experience	-644.6	644.4	-1.0	0.31

Results of the Wilcoxon test on Find Time (FT_o) are presented next (Row 1 of Table 9.6). Considering all camel-cased and underscored identifiers (global measure), a significant difference ($p\text{-value}=0.01$) in Find Time is noted, with camel-cased identifiers taking longer overall. The effect size is moderate (Cohen's $d=0.57$), which is considered to be practically significant. Taking a closer look, we find this significance only in the 3-word phrases. No significance is detected within the 2-word code/non-code identifiers, although camel-cased identifiers take longer on average. We also note that the read time (RT_p) of phrases used for each identifier style follow the same distribution.

9.5.2 Visual Effort

Visual Effort is measured by the six measures defined in Section 9.4.3. Three measures relate to the number of fixations and the rest relate to the time involved in those fixations. Results of the Wilcoxon test for each of these measures are given in Table 9.6. We now consider each separately.

**Table 9.6. Two-tailed Wilcoxon p -values ($\alpha=0.05$) for each dependent variable.
CC=camel case, US=underscore**

Dependent Variable	Grouped by Style	Grouped by Style and Phrase Length		Grouped by Style, Phrase Length and Origin			
	CC vs. US (Cohen's d)	2-word ident. (code $\cup$ non code)	3-word ident. (code $\cup$ non code)	2-word code identifiers	3-word code identifiers	2-word non code identifiers	3-word non code identifiers
FT_Q	0.01 * (0.57)	0.437	0.007 *	0.772	0.009 *	0.366	0.05 (~*)
FC_Q	0.213 (0.24)	0.029 *	0.033 *	0.380	0.057 (~*)	0.004 *	0.124
FR_{correct}	0.599 (0.15)	0.277	0.720	0.561	0.720	0.699	0.561
FR_{distracters}	0.599 (0.15)	0.277	0.720	0.561	0.720	0.699	0.561
AFD_Q	0.008* (0.21)	0.151	0.004 *	0.890	0.041 *	0.058 (~*)	0.277
AFD_{correct}	0.015* (0.33)	0.04 *	0.208	0.679	0.277	0.016 *	0.543
AFD_{distracters}	0.026* 0.21	0.064	0.489	0.639	0.169	0.075	0.524

With respect to the fixation count (FC_Q) and fixation rate (FR_{correct} and FR_{distracters}) no significant difference was found between identifier styles with respect to the entire data set grouped by identifier style (p -values=0.213, 0.599, 0.599: rows 2 through 4 in Table 9.6).

For FC_Q (total number of fixations on the question stimulus) grouping identifiers by phrase length does give a significant improvement favoring underscore identifiers for both 2-word (p -value=0.029) and 3-word (p -value=0.033) identifiers, suggestive of a larger number of fixations for camel-cased identifiers. On average, there are six more fixations on 3-word camel-cased identifiers (i.e., *extendAliasTable*, *movieTheaterTicket*)

compared to 3-word underscored identifiers (i.e., *get_next_path*, *read_bedtime_story*). The 2-word identifiers differ by only two fixations.

Breaking down the categories even further shows significance only for 2-word non-code identifiers ($p\text{-value}=0.004$), with 3-word code identifiers approaching significance ($p\text{-value} = 0.057$). The rate of fixations on correct and incorrect identifiers (FR_{correct} and $FR_{\text{distracters}}$) shows no significant difference globally or in any phrase category. This suggests that the number of fixations needed between the two identifier styles is not very different for both correct identifiers and the distracters.

With respect to the average fixation duration (AFD_0 , AFD_{correct} , and $AFD_{\text{distracters}}$) there is a significant difference between identifier styles over the entire data set ($p\text{-values}=0.008, 0.015, 0.026$: rows 5 through 7 in Table 9.6). In particular, for the question stimuli (AFD_0), 3-word code identifiers are statistically significant ($p\text{-value} = 0.041$). The distribution shows camel-cased identifiers require a higher average duration of fixations. Figure 9.4 shows the gaze plot for two 3-word code identifiers showing a larger number and increased duration of fixations for the camel-case style. A fixation is shown as a circle, whose radius represents the duration.

Figure 9.4. Part of two gaze plots for the correct underscore and camel cased versions of the 3-word code identifier

Figure 9.5. Descriptive statistics for the Average Fixation Duration on correct identifiers ($AFD_{correct}$)

The average fixation duration of correct identifiers ($AFD_{correct}$) is significant at the 2-word phrase ($p\text{-value} = 0.04$) and in particular for non-code identifiers ($p\text{-value} = 0.016$). See Figure 9.5 for the distribution. There was no statistical significance with respect to $AFD_{distracters}$ within any grouping of identifiers, except for the global measure that considers all camel-cased and underscored identifiers together. Overall, based on the distribution, this suggests that for camel-cased identifiers; time taken to read the distracters is more than the underscored identifiers.

Figure 9.6 shows part of a gaze plot depicting a distracter at the beginning of a 3-word non-code identifier (*mouseTheaterTicket*). Three large fixations are seen at the beginning or middle of each part (*mouse*, *Theater*, and *Ticket*) of the compound word. This indicates a longer mental parsing time needed to process the joined word.

Figure 9.6. Part of a gaze plot for a novice showing three large fixation durations on each of the three parts of the distracter *mouseTheaterTicket*

The AFD_0 mimics the FT_0 measure in terms of significance in 3-word identifiers, whereas both $AFD_{correct}$ and $AFD_{distracters}$ also show significance for the 2-word identifiers but not in the 3-word category. Based on the above measures, overall, we can reject $H5_0$, in which case we accept the alternative hypothesis that visual effort is affected by identifier style. Although fixation count/rate is not significant on its own, the average fixation duration uses the fixation count along with the time of each fixation, together having a significant effect on time required to comprehend identifiers.

Finally, after testing for interactions using *ANOVA*, we have to reject the null hypothesis $H6_0$ since Style does not significantly interact with any secondary variable to have an affect on the dependent variables.

9.5.3 Object Memory Task

After the main task of finding the eight identifiers was complete, subjects did a simple object memory task to detect any differences in short term memory between the two identifier styles used. This involved studying a short C++ code snippet, comprised of two methods, each 12-15 lines long, for as long as they needed. Next, a set of nine

identifiers were presented and they were asked to choose which identifiers exactly matched the ones in the code snippet. There were four correct choices (two camel case and two underscore), with the rest being distracters that changed the style or letters in the identifier. None of the subjects gave completely correct answers. On average, one of each camel-case and underscore identifier was recalled correctly but there were several false positives. This small exercise suggests no difference in recall between the two types of identifier style. This task was also conducted using the eye tracker, however eye gaze analysis of the code is not presented here.

9.5.4 Similarities and Differences

The goal of this study was the same as the Binkley's study. Both studies use the same set of phrases to test for differences in identifier styles. The main difference between this and the previous study is the method of data collection. An eye tracker is used in this study. Conditions were also more strictly controlled with no additional personal delay biases, since the moderator advanced the screen as soon as the subject verbally stated the answer. This study was conducted as a within-subjects design where all subjects are exposed to all treatments of the factor and pair wise comparisons are made between each identifier style. With respect to the question stimuli, the clouds are not animated as in the previous study and the phrase to be studied is not included on the question stimuli.

Another difference is the historical training received by the subjects. In this study, subjects were trained primarily in the underscore style and they were all programmers unlike the original study. An Experience variable replaced the Training

variable from the original study, due to an all programmer subject sample. With respect to analyzing results, in addition to the linear mixed-model analysis (common to both studies), data is also analyzed using the non-parametric Wilcoxon test within each category due to non-normality of certain identifier groups and low sample size. In addition, an object location task is added as a post-task in this study to determine the recall ability of subjects with respect to camel-cased and underscore identifiers.

9.6 Discussion

In response to the research questions posed in Section 9.3, we find that identifier style significantly affects time and visual effort needed to correctly detect identifiers constructed from a phrase. The underscore style is significantly faster and positively influences the dependent variables. In the Binkley study Phrase Length did not interact with Style, however we find such an interaction in our analysis. The common theme in both experiments is that camel-case takes longer than underscore (13.5% in the previous study and 20% in this study) overall. In our study however, no interaction effects were found between the Experience secondary factor and the independent variable Style. The previous study found Training (vis-à-vis Experience in our study) to significantly interact with Style affecting the time to find an identifier. Their findings indicate that subjects trained in the camel-case style take less time to identify a camel-cased identifier than an underscore identifier.

We observed that the difference between Expertise among subjects seems to interact with Style and have an effect (albeit not significant) on Find Time in two ways. First, the underscore style reduces the gap between expert and novice performance

(Figure 9.3) and second, novices seem to benefit approximately twice as much from the underscore style than experts. Comparing this result to the original study, we can say that with more experience (training), the effect of identifier style on performance is reduced, but not eliminated.

In the Binkley study, only one-third of the subjects were trained in camel-cased identifiers. In our case, we have an equal proportion of experts (8) and novices (7). During the demographic briefing, six subjects (40%) stated that they preferred camel case, seven (47%) stated that they preferred underscore, and two (13%) had no preference. Also, in the Binkley study, non-programmers stated that camel casing would be harder to visually process and thus lead to more errors. Our results prove this claim to be true with the data generated from the eye tracker (AFD_Q , $AFD_{correct}$, $AFD_{distracters}$).

We did not look at eye fixation order, sequence of moving from one cloud to another, and the number of regressions involved due to the relatively simple nature of the task. These measures would be more pronounced with reading a block of code versus just identifiers.

9.7 Threats to Validity

Internal validity refers to the presence of other factors besides the main factor that might have an effect on the results. Since this was a within-subjects experiment we had to make sure that there was no learning effect involved when comparing the results of the two treatments for a particular phrase type. We address this by using a similar but different phrase for each identifier style. Question order was set randomly and then fixed for each subject. Another threat to validity is the type of reading behavior subjects

engaged in. Reading from top to bottom versus left to right might have an impact on how quickly subjects find an identifier. After analyzing the gaze plots for each subject, we find that almost all subjects look at each cloud to determine their final answer. In this case, the fixations and time are spread out evenly.

External validity deals with generalizing our results to a real-life setting. We used students as subjects in our study, however the novices are comparable to junior software developers and experts are comparable to senior level developers since all of them have extensive programming experience. The number of our subjects appears to be low, however eye-tracking studies usually have about the same number of subjects [Goldberg, Kotval 1999]. The nature of the task used is not typical of reading code however; the basic reading process is still the same.

Construct validity refers to the validity of the measures used to measure performance. Since visual attention is related to mental processing of the information [Just, Carpenter 1980], the measures derived from the fixation counts and durations should be valid.

To ensure conclusion validity, we use the non-parametric paired Wilcoxon statistical test to determine significance due to non-normality of certain dependent variables in certain identifier groups and also less importantly due to low sample size. *ANOVA* is used for normally distributed samples to determine interaction effects, if they exist.

9.8 Prior Work

This section presents existing work on identifier names, source code readability and quality, and psychology research on reading. Relevant eye tracking research related to code and diagrams is already discussed in Section 2.4 and is not repeated here.

9.8.1 Identifier Names

Lawrie et al. [Lawrie, Morrell, Feild, Binkley 2006] conduct a large study on identifier names and show that actual words rather than abbreviations lead to better comprehension. Butler et al. [Butler, Wermelinger, Sharp 2009] study the effect of identifier names on the quality of code. They find that identifiers that violate certain guidelines have lower code quality (more bug patterns) than ones that don't. Caprile et al. [Caprile, Tonella 2000] study the restructuring of identifier names and the arrangement of individual words in identifiers. Binkley et al. [Binkley, Lawrie, Maex, Morrell 2009] study the effect of identifier length on the recall ability of programmers, showing that longer names reduce correctness and take longer to recall. Our results in this paper add to this finding, since phrase length significantly interacts with identifier style to have an effect on performance. None of the above work considers the effect identifier style has on comprehension with the exception of [Binkley, Davis, Lawrie, Morrell 2009]. The research presented here nicely complements these approaches for better identifier names.

9.8.2 Research on Reading

In psychology research, Epelboim et al. [Epelboim et al. 1997] conducted a study on the effect fillers have on reading time. Spaces between words are filled with different fillers: Latin and Greek letters, digits and shaded boxes. They found that the type of filler had a significant effect of slowing reading speed anywhere between 10-75% depending on the filler. Shaded boxes between words (similar to underscores) had the smallest effect on reading time. Rayner et al. [Rayner, Fischer, Pollatsek 1998] also show a decrease in reading rate by approximately 50% when fillers like *x* were used between words. Our results in this study support the above findings since a significant improvement in Find Time for underscores is shown.

9.9 Summary and Future Work

An eye-tracking study analyzing the effect of identifier style (camel-case and underscore) on accuracy, time, and visual effort is presented with respect to the task of recognizing a correct identifier, given a phrase. Visual effort is determined using six measures based on eye gaze data namely: fixation counts and durations. Although, no difference was found between identifier styles with respect to accuracy, results indicate a significant improvement in time and lower visual effort with the underscore identifier style. The interaction of experience with style indicates that novices benefit twice as much with respect to time, with the underscore style. This implies that with experience or training, the performance difference between styles is minimized. These results add to the findings of the Binkley study [Binkley, Davis, Lawrie, Morrell 2009]. Future work includes conducting more eye-tracking studies (with a larger subset of identifiers and

larger subject sample), on reading source code consisting of both identifier styles, in the context of solving a specific task.

CHAPTER 10

Guidelines and Statements based on Empirical Evidence

This chapter summarizes a list of thoughts generated from the experiments conducted. First, an overview of the experiments is given followed by guidelines and statements that are direct consequence from the data acquired off the experiments.

An overview of the experiments conducted in this dissertation is given in Table 10.1.

Table 10.1. An overview of the experiments conducted

Experiment	Unique Diagrams	Layouts	Total Diagrams	Subjects	System	Number of classes	Questions
Pilot study (Chapter 4)	5	3	15	20	Hippodraw	50	27
Yusuf et al. replication study (Chapter 5)	6	3	18	29	Hippodraw	100	27
Task categories study (Chapter 6)	16	2	32	45	Qt wxWidgets	122 109	24
Design pattern role detection study via questionnaires (Chapter 7)	8	2	16	17	JHotdraw JUnit Qt	49 27 36	16
Design pattern role detection study via an eye tracker (Chapter 8)	8	2	16	15	JHotdraw JUnit Qt	49 27 36	16
Totals	43		81	126	5		94

A total of five experiments are conducted, each of which is described in a separate chapter. A total of eighty-one diagrams were analyzed over one-hundred-and-twenty-six

subjects. Since the diagrams in Chapter 8 are the same as Chapter 7, this number is not double counted in the total number of diagrams used. However, the unique diagrams are added since the layouts were flipped for this replicated study. There is a small overlap (~9) in subjects across a few of the studies. These studies were conducted over a period of two years. A total of five systems were analyzed over ninety-four questions. Some of these questions were grouped into task categories as is the case with the studies in the first three rows of the table. The systems were chosen from different domains and languages. They were drawing applications (JHotDraw, Java), UI frameworks (Qt, wxWidgets, C++), data analysis environment package (Hippodraw, C++) as well as testing frameworks (JUnit, Java).

The next part of this chapter presents some discussion on the guidelines to be followed while adjusting the layout of class diagrams as well as lessons learned. These guidelines should be taken into account while constructing the layout adjustment algorithm. These statements have been collected throughout the experimental process and are directly based on the results of the experiments done, rather than on anecdotal evidence.

- Determine the task to be solved. The layout depends on the task. This is evident in the task categories study in Chapter 6. Certain tasks might benefit from the three-cluster layout (shown in Chapter 4), whereas the multi-cluster layout was useful for more challenging tasks. In some cases, the orthogonal layout was sufficient and did as good as the multi-cluster layout. However, we did not find a case where the orthogonal layout was better than the multi-cluster layout.

- Proximity is the main factor that affects comprehension, not pretty aesthetics. Proximity may be defined differently based on the task. One definition of proximity is the level of interrelationships between classes. In this case, proximity is related to the physical connections (relationships between classes). Another definition could be dependent on the types of methods and attributes of each class. It may also be based on certain metrics or heuristics defined for each class.
- Classes that work together should be placed closer together even though they might violate some aesthetic criteria. By work together, we mean classes that are related to each other in the form of associations, aggregations, compositions, or dependencies. This is the physical proximity referred to in the previous point and is the main factor investigated in the dissertation.
- If a class A in a generalization hierarchy works closely with a class B not in the same hierarchy, then class A should be placed closer to class B even though this might violate the constraint of the generalization hierarchy. The generalization hierarchy aesthetic states that all children in the hierarchy should be placed below the parent and in the same horizontal level based on the level of the hierarchy. In all the studies conducted here, the multi-cluster layout tends to violate this constraint without a negative impact on comprehension. An excellent example is the role detection study (Chapter 7 and Chapter 8), where the classes in the Strategy pattern are placed closer together in the multi-cluster layout even though the class with the role of Context might be part of another hierarchy.

- Avoiding edge crossings and bends are not as important as they are claimed to be in previous studies. Adding a few extra edge crossings or bends in a diagram does not affect the performance especially if these additions cause related classes to be placed together in a cluster.
- Software maintainers or designers who are the intended audience of the stereotyped layouts need to be trained to use information about stereotypes before the layout approach can help them to the full potential. For example, training users to use class stereotypes to solve maintenance tasks would increase the usefulness of the multi-cluster layout.
- The layout approach of architectural importance is applicable to any system and hence the approach will work for any object-oriented system.
- Class diagrams need not represent all relationships between the classes to be useful. A filtering mechanism should be used where only relevant priority relationships are displayed. The class diagram may represent an incomplete though useful picture of the system.
- The multi-cluster layout can be augmented by overlaying different types of class or method level metrics on top of each class.
- A diagram should not have more than four or five (at most) clusters. This also sets a limit on how many classes should be shown in a diagram. A cluster should not include more than four or five classes. A few more may be included if the attributes and methods of the classes are hidden.

- Within each cluster, classes and relationships may follow an orthogonal layout. This would be considered local to the cluster.
- The layout should be customizable and interactive based on the task. In this case, the number of classes and clusters can be interactively collapsed or expanded as needed keeping in mind the set limit of showing no more than five clusters at once.
- People generally look at the top of the diagram and the center of a diagram. The lower right part of a diagram is very rarely looked at. This tells us to place important classes in the system towards the top half of the page.
- Experts differ from novices in their eye gaze and fixation order. They tend to scan quickly and find the relevant classes in a cluster if the layout places them closer together. These clusters act like beacons for easy recognition.
- Experts tend to find relevant clusters quickly and then focus their visual effort in a cluster. Novices also benefit from the multi-cluster layout and their performance gets close to an experts. However, the difference between experts and novices is much larger in the orthogonal layout.
- The clusters in the multi-cluster layout act like design beacons that quickly highlight classes participating in a certain functionality of the system. This is evident from the eye-tracking study on design pattern role detection.
- It is important to use both fixation counts and gaze durations to determine the effort since based on our results we find that even though fixation counts may not be significantly different, the duration is significant, and vice versa.

- Eye fixations are not commonly found on relationship lines, however they are very commonly found on relationship ends. This is due to the quick nature of eye movements. Hence, it is important to not only include class bounding boxes in the visual effort measures but also fixations and durations on the relationship ends. Without this information the effort will not be accurate.

CHAPTER 11

Conclusions and Future Work

The dissertation presents a family of experiments that examine the effects of different layout techniques for Unified Modeling Language (UML) class diagrams. Three different layout schemes are examined. The architectural importance of classes is used to guide each layout technique. Architectural importance in a UML class diagram is defined by stereotype information, in this case the control, boundary and entity class stereotypes. The main argument is that organizing a class diagram based on architectural importance rather than abstract graph guidelines, increases system comprehensibility.

The experiments use traditional questionnaire-based methods as well as eye-tracking equipment to quantitatively measure the performance of subjects solving specific software maintenance tasks. Different levels and categories of software engineering tasks are examined. The layout techniques are also applied to design pattern comprehension in class diagrams. Two of the experiments are replicated using an alternate method of data collection to support and verify the findings. This replication process further validates and deepens the knowledge acquired about the layouts involved in the tasks studied. A total of five experiments are conducted with a total of eighty-one diagrams using ninety-four questions in five real software systems. Two of the systems are written in Java, while the other three are C++ systems. The domains of the systems were not homogeneous. In addition to UML class diagram layouts, the effect of identifier styles

(camel case and underscore) on the readability of UML class diagrams is also examined and is viewed as complementary to layout schemes used.

The main contribution of the dissertation is to the evidence-based software engineering field which states the importance of empirical evidence to prove usefulness of an approach. The one sure way of determining the usefulness of diagram layouts is to conduct empirical studies to determine the usefulness with users who are intended to use the diagram for real tasks. The empirical evidence presented states that layout significantly impacts comprehension. In particular, the multi-cluster layout proved to be the best layout overall, especially for more challenging tasks and had a positive effect on accuracy, speed and visual effort needed to solve the tasks. The groups or clusters in the multi-cluster layouts act as visual beacons analogous to beacons present in source code [Brooks 1983]. When these visual beacons are present in the class diagram they tend to reduce cognitive load and effort to solve the task. This premise is validated using quantitative eye-tracking data. The measures for visual effort given in the dissertation provide an objective metric to measure the quality of UML class diagram layouts in practice.

The results of this work directly impact both industry and academics. In academia, better layouts can be used to improve the teaching of design patterns. When class diagrams and design patterns are introduced to students, they should also be made aware of how layout can have an impact on comprehension. The focus should be on aesthetics and comprehensibility. In industry, using a good layout will help in accurately identifying the correct solution in less time and effort. If layout reduces the initial effort

a software maintainer needs to go through in order to understand the diagram, more time can be spent on solving the task rather than worrying about correctly tracing a relationship between classes or even failing to see something important due to the nature of the layout. The goal is making the UML class diagram more accessible to the maintainer of the system.

An immediate future goal is to realize these empirical results into a tool to adjust the layout of a UML class diagram for specific tasks. The guidelines and statements presented here will directly affect the algorithm used in adjusting the layout. Another research direction involves conducting more studies, using questionnaires and eye-tracking equipment, on other software engineering tasks with respect to class diagram layouts as well as source code. For example, a study is being constructed that compares two layouts for detecting defects in UML class diagrams with respect to a requirements specification. The task here is to identify defects in the design document, namely the class diagram and is directly related to software inspections.

Another study would be to conduct a videotaped session of subjects adjusting the layout of a class diagram while given a maintenance task. This study could uncover many unspoken behaviors that might aid in the understanding of the way users would like a diagram to be laid out. With respect to studying design patterns, the plan is to include other design patterns such as Visitor, Proxy, and Bridge and determine the effect layout has in more complex tasks such as adding specific functionality to the system. Another direction is to investigate the relationship of class stereotypes in design pattern role detection with respect to maintenance tasks. Future investigations also include analyzing

the effect of different levels of detail shown in class diagrams (such as hiding of attributes and methods) and its interaction with layout on performance. The study on identifier styles can also be expanded to include the different identifier styles within class diagrams as well as in real code snippets and determine the effect it has on performance in certain software maintenance tasks such as software debugging.

APPENDIX A

Related Work Overview

Table 11.1. UML class diagram comprehension studies closely related to the research

Researchers	Goal	Layout Aware?	Stereotypes Used	Empirical Investigation	Method of Data collection
[Andriyevska, Dragan, Simoes, Maletic 2005]	Evaluate effectiveness of 3 layouts based on stereotypes.	Yes	Control Boundary Entity class stereotypes	Controlled experiment 20 graduate students	Online Questionnaire
[Yusuf, Kagdi, Maletic 2007]	Identify factors most effective in supporting comprehension and understand how humans look at class diagrams	Yes	Control Boundary Entity class stereotypes	Controlled experiment 8 graduate students 1 undergraduate student	Tobii 1750 eye tracker
[Kuzniarz, Staron, Wohlin 2004; Staron, Kuzniarz, Wohlin 2006]	Determine if stereotypes helped in model understanding	No	Telecommunication stereotypes via graphical icons (sender receiver and transmitter)	Controlled experiment 44 subjects	Questionnaire
[Staron, Kuzniarz, Thurn 2005]	Determine if stereotypes helped in finding faults	No	Telecommunication stereotypes via graphical icons (sender receiver and transmitter)	Controlled experiment 11 subjects	Questionnaire
[Ricca et al. 2006]	Determine if stereotypes contribute to understanding web applications	No	Conallen's stereotypes (web application and extension)	Controlled experiment 35 undergraduate students	Questionnaire

Researchers	Goal	Layout Aware?	Stereotypes Used	Empirical Investigatio n	Method of Data collection
[Ricca et al. 2007]	Determine if subjects' ability and experience support the use of stereotypes.	No	Conallen's stereotypes (web application and extension)	Controlled experiment 35 undergraduate students 13 graduates @ University X 18 graduates @ University Y	Questionnaire
[Guéhéneuc 2006]	Determine how software engineers identify design information from a class diagram.	No	None	Controlled experiment 12 graduate students	Head mounted eye tracker
[Jeanmart, Guéhéneuc, Sahraoui, Habra 2009]	Determine effect of Visitor pattern on program comprehension	Yes	None	Controlled experiment 24 subjects	Head mounted eye tracker

APPENDIX B

Study Material for Chapter 4

B.1. Comprehension Questions

The comprehension questions for the pilot study are given below. The X in the Diagram shown column represents one of three possible layouts. Each of the three groups were presented with one of the three possible layouts.

Comprehension Questions	Diagram Shown
1. Which class from the following class diagram is responsible for drawing to a display device?	D2_X
2. Which class is responsible for creating and maintaining model for axes (i.e. draws axes, titles, labels but not data points):	D2_X
3. Which class is an interface between the GUI and DataSource objects?	D2_X
4. Which class is responsible for storing column data?	D2_X
5. Which <i>abstract class</i> is responsible for raw data and acts as a base class to a class storing column data?	D2_X
6. Which class is responsible for creating projected values from raw data?	D2_X
7. Which class is responsible for representing projected values graphically?	D2_X
8. Which of the choices given below best describes the role of the <i>DataSource</i> class in HippoDraw?	D3_X
9. Which of the choices given below best describes the role of the <i>DataSourceController</i> class in HippoDraw?	D3_X
10. We would like to add a method to the plotting feature in HippoDraw. The method deals with drawing a colored square	D3_X

<p>mainly used for shading in gray plots. The method takes x, y coordinates and a color as parameters. The signature of this method is</p> <pre>void drawSquare (double x1, double y1, double x2, double y2, int red, int green, int blue)</pre> <p>In which class would you most likely add it?</p>	
11. You just found out that the statistical features implemented in HippoDraw do not help you analyze data sets using the difference between size distributions. You know that the <i>Kolomogorov-Smirnov</i> statistical test is what you would need to implement to analyze the data points from <i>NTuple</i> in order to carry out your analysis.	D3_X
Which class should this feature be derived from?	
12. Add a method <i>registerDataSourceFile</i> that registers that the DataSource came from a file. The signature of this method is	D3_X
<pre>void registerDataSourceFile (DataSource * ds).</pre> <p>In which class would you add it?</p>	
13. Add a method <i>findDataSource</i> that returns the <i>DataSource</i> object with registered name <i>name</i> and if an object of the given name is found, returns a pointer to it, otherwise throws an exception. The signature of this method is	D3_X
<pre>DataSource * findDataSource(const std::string & name) const</pre> <pre>throw (DataSourceException)</pre> <p>In which class would you add it?</p>	
14. Which of the choices given below best describes the role of the <i>PlotterBase</i> class in HippoDraw?	D4_X
15. Which of the choices given below best describes the role of the <i>ViewBase</i> class in HippoDraw?	D4_X
16. We want to add a feature, plotting in 3-D, which would plot objects in actual 3 dimensions. Which class do you think this feature should be derived from? Plan for anticipated changes rather than immediate fixes.	D4_X
17. One of the features requested and currently in the Todo list is to be able to increase the size of labels if they are hard to read in a plot. You decide that you would like to add a method to a class that performs this task. In which class would you add it?	D4_X
18. Another feature requested is to perform spell check on labels and any other text that exists on plots. Which is the most likely action you would take?	D4_X
19. Which of the choices given below best describes the role of the	D5_X

DataRep class in HippoDraw?

20. Do you observe a *Design Pattern* in the given class diagram? If yes, then briefly name and describe *all* the main classes you think are involved in this pattern
-

B.2. Preference Questions

21. Regarding HippoDraw, select all that apply. You can choose multiple.
- a. I worked with HippoDraw code and am very familiar with their design including major relationships and hierarchies
 - b. I used HippoDraw only as an end user to plot data
 - c. I reverse engineered HippoDraw and have a basic understanding of relationships that exist
 - d. Today is the first time I am hearing of HippoDraw
 - e. Other (explain if not any of the above)
22. How do you rate your knowledge regarding HippoDraw?
- a. Excellent
 - b. Somewhat Familiar
 - c. I am seeing HippoDraw for the very first time today
23. Select the most appropriate regarding class stereotypes and explain if needed
- a. The stereotype annotations of entity, boundary and control as well as the colors used helped me answer the questions
 - b. The class stereotypes could be used more efficiently in terms of layout (give reasons below)
 - c. The class stereotypes marked with text and in different colors did not play any role in answering questions
 - d. The class stereotypes marked with text and different colors did not help me to answer questions but in fact caused problems and confusion in understanding the system
24. Based on colors used, choose the appropriate option below
- a. The colors used in defining entity, boundary and control classes were terrible and were hindering in answering questions
 - b. I liked the colors
 - c. I don't care too much for the colors
25. This question compares two diagrams, one that uses orthogonal lines between classes and another that uses curved connectors for relationships other than generalizations. Both the diagrams are shown below. Choose the most appropriate option keeping in mind if either diagram would cause a hinderance in understanding or browsing the system. Do not consider aesthetics at this point.
- a. The difference between the curved and orthogonal connectors and how they were used quickly brought my attention to certain aspects of the diagram and would possibly help me answer a question faster
 - b. The use of curved and orthogonal connectors would not play a role in understanding the system

- c. I found it difficult to understand the diagram that used curved connectors for associations and dependencies
- d. Both look reasonably good to me
- e. I have a suggestion to use curved and orthogonal connector and it is stated below:
- f. The diagram with curved connectors looks weird

Based on the same two diagrams above, choose the most appropriate choices only basing your decision on how the diagrams look (aesthetics). Try not to contradict your choices.

- a. The diagram with the curved connectors was pleasing to the eye and looked nice
- b. The diagram with the curved connectors was not aesthetically pleasing
- c. The diagram with the orthogonal connectors was pleasing to the eye instead of the curved connectors
- d. The diagram with the orthogonal connectors was not aesthetically pleasing
- e. Both diagrams were not very aesthetically pleasing to the eye
- f. Both diagrams were aesthetically pleasing to the eye

Diagrams Used: d3 curved lines in all three layouts

26. Would you prefer to see a class diagram with certain cohesive areas enclosed in a light shading like the diagram below?
- a. Yes it is much better in understanding small parts of the class diagram and progressively moving through the whole diagram through clusters to build a better mental model
 - b. No I do not like the shading that encloses clusters of classes together

27. Three Class Diagrams with the same number of classes and relationships are shown below. Each named *Diagram 1*, *Diagram 2* and *Diagram 3* respectively. Answer two questions about how you would go about comparing these two diagrams below. (hippodraw overview with attributes and ops D2) Which diagram would you look at if you were interested in understanding the system and not paying too much attention on layout? (An edge crossing would not matter so much, for e.g.) *

- a. Diagram 1
- b. Diagram 2
- c. Diagram 3

Which diagram did you like the best in terms of aesthetic layout?*

- a. Diagram 1
- b. Diagram 2
- c. Diagram 3

Diagrams Used: D2 in all three layouts

28. Three Class Diagrams with the same number of classes and relationships are shown below. Each named *Diagram 1*, *Diagram 2* and *Diagram 3*

respectively. Answer two questions about how you would go about comparing these two diagrams below. (plotterbase detailed view D4)

Which diagram would you look at if you were interested in understanding the system and not paying too much attention on layout? (An edge crossing would not matter so much, for e.g.) *

- a. Diagram 1
- b. Diagram 2
- c. Diagram 3

Which diagram did you like the best in terms of aesthetic layout?*

- a. Diagram 1
- b. Diagram 2
- c. Diagram 3

Diagrams Used: D4 in all three layouts

29. Any comments about this user study are welcome. Any suggestions, comments as well as critiques are more than welcome.

B.3. Diagrams in Orthogonal Layout

HippoDraw Overview [D1]

DataSource Detailed View [D3]

PlotterBase Detailed View [D4]

Data Representation Detailed View [D5]

HippoDraw Overview with Attributes and Operations [D2]

DataSource Detailed View [D3]

PlotterBase Detailed View [D4]

Data Representation Detailed View [D5]

B.5. Diagrams in Three-cluster Layout

HippoDraw Overview with Attributes and Operations [D2]

APPENDIX C

Study Material for Chapter 5

C.1. UML Notation Questions

The questions and diagrams from the replication of Yusuf et al.'s study are given. The diagram ID's compared are (S1, S2 and S3), (S4, S5, S6), (S7, S8, S9) and (S10, S11, S12).

UML Questions	Time allowed	Correct Answer	Score	Layout Type	Diagram ID
1. Identify the kind of relationship between <i>ViewBase</i> and <i>PlotterBase</i> .	50 sec	Dependency	1	Orthogonal	S1
2. Select all the classes involved in aggregation.	60 sec	PyApp Hdthread PyDataRep DataRep	1-4	Orthogonal	S4
3. Select all the derived classes of <i>PlotterBase</i> .	60 sec	CompositePlot TextPlotter XyPlotter CutPlotter Cut1DPlotter	1-5	Orthogonal	S7
4. Which class has a method named <i>getAverage</i> ?	50 sec	NTupleProjec	1	Orthogonal	S10
5. Identify the kind of relationship between <i>NTuple</i> and <i>DataSource</i> .	50 sec	Aggregation	1	3 cluster	S2

6.	Select all the classes involved in dependency.	60 sec	NTupleCont. NTuple PyFuncRep PlotterBase PyDataSource DataSource	1-6	3 cluster	S5
7.	Which classes are derived from <i>Observer</i> ?	60 sec	ViewBase DataRep PlotterBase	1-3	3 cluster	S8
8.	Which class has a method named <i>objectiveValue</i> ?	50 sec	NTupleChiSqF	1	3 cluster	S11
9.	Identify the kind of relationship between <i>DataSource</i> and <i>Observable</i> .	50 sec	Generalization	1	Multi cluster	S3
10.	Select all the classes involved in generalization.	60 sec	Observable DataSrc NTuple PyNTuple DataRep PlotterBase QThread HdThread FunctionRep	1-9	Multi cluster	S6
11.	Which classes are involved in aggregation?	60 sec	AxisModelBase AxisTick Observable Observer Range TupleCut	1-6	Multi cluster	S9
12.	Which class has a method named <i>registerNtuple</i> ?	50 sec	NTupleControl	1	Multi cluster	S12

C.2. High-Level Design Questions

The diagram ID's compared are (S13, S14 and S15), (S16, S17, S18), (S19, S20, S21), (S22, S23, S24), and (S25, S26, S27).

Design Questions	Time allowed	Correct Answer	Score	Layout Type	Diagram ID
13. Select the class that a python wrapper uses to access data in the class <i>NTuple</i> .	60 secs	PyNTuple	1	Orthogonal	S13
14. Select the class responsible for managing XML serialization.	60 secs	XmlController	1	Orthogonal	S16
15. Which class controls the active window of an application?	60 secs	WindowController	1	Orthogonal	S19
16. Select the base class for the axis representation hierarchy?	60 secs	AxisRepBase	1	Orthogonal	S22
17. Select the class through which a boundary class could access data in the class <i>NTuple</i> .	60 secs	NTupleControlr	1	Orthogonal	S25
18. Select the class that is a python wrapper for a	60 secs	PyFunctionRep	1	Three-cluster	S14

class with the method name <i>addcut</i> .						
19. Which classes are specialized for XML processing in QT.	60 secs	QtXmlElement QtXmlNode QtXmlTextNod	1-3	Three-cluster	S17	
20. Which class responds to toolbar events from windows and messages sent by the class <i>Inspector</i> ?	60 secs	CanvasWindow	1	Three-cluster	S20	
21. Select the class that plots a point in 2D.	60 secs	XyPlotter	1	Three-cluster	S23	
22. Name the class through which a boundary class could access data in the class <i>DataSource</i> .	60 secs	DataSrcControlr	1	Three-cluster	S26	
23. Which entity class is responsible for storing data?	60 secs	NTuple	1	Multi-cluster	S15	
24. Select the entity class that could be extended to specify a new property (besides <i>Font</i> and <i>Color</i>) in XML	60 secs	BaseXML	1	Multi-cluster	S18	

25. Name the concrete class that displays data in a tabular format.	60 secs	PlotTable	1	Multi-cluster	S21
26. Which class sets the range and scale of the axis?	60 secs	AxisModelBase	1	Multi-cluster	S24
27. Select the classes that get data from the class <i>DataSource</i> objects and use functions from the class <i>FunctionBase</i> .	60 secs	NTupleLikeHood NTupleChiSFC	1-2	Multi-cluster	S27

S4

S10

S13

S16

S19

S22

S25

C.4. Diagrams in Multi-cluster Layout

S3

S6

S9

S12

S15

S18

S21

S24

C.5. Diagrams in Three-cluster Layout

S2

S5

S8

S11

S14

S17

S20

S23

S26

APPENDIX D

Study Material for Chapter 6

D.1. Comprehension Questions for the Qt system

Questions on the Qt system	Diagrams Shown ⁷
1. What is the class stereotype assigned to <i>QSqlDatabase</i> ?	qt-D1 qt-D2
2. Which classes are dependent on <i>QPainter</i> ?	qt-G1 qt-G2
3. Is this statement true or false? The control class <i>QDragManager</i> is associated with the entity class <i>QPixmap</i> via the attribute <i>pm_cursor</i> .	qt-I1 qt-I2
4. Which class is involved in a composition relationship with <i>QAction</i> and is part of the <i>QEvent</i> hierarchy?	qt-E1 qt-E2
5. Qt's model/view architecture makes it possible to display the same data in different ways without changing the underlying data structure, delivering flexibility and scalability to large data sets. Which concrete class (or classes) implement the view component in the model/view architecture? Select all that apply.	qt-MV1 qt-MV2
6. Select the class responsible for setting the central widget of a window.	qt-MW1 qt-MW2
7. Which abstract class is responsible for managing layout for the graphical user interface?	qt-L1 qt-L2
8. A view renders the contents of a model. You need to use a model with <i>QListView</i> . Which class would you use to represent the model?	qt-MV1 qt-MV2
9. SVG is an XML-based file format and language for describing 2	qt-G1

⁷ One of the two diagrams were shown for each question. The first one is in multi-cluster layout, the second is in orthogonal layout

dimensional graphics.	qt-G2
Which class controls the drawing of SVG files to a paint device?	
10. Which class provides an action that can be inserted into widgets?	qt-W1 qt-W2
11. Which concrete layout related classes and widget related classes are used in the Dialog box below? Select all layout related concrete classes and widget related concrete classes that apply.	qt-L1 qt-L2

12. Which input event class stores information about which key on the keyboard was pressed?	qt-E1 qt-E2
13. Qt allows data obtained from models to be related to specific widgets. Which class is responsible for setting up this data relation between SQL models and widgets?	qt-D1 qt-D2
14. Consider the Analog clock QT application shown below. The main program is shown below.	qt-MW1 qt-MW2
<pre>#include < QApplication > #include "analogclock.h" int main(int argc, char *argv[]) { QApplication app(argc, argv); AnalogClock clock; clock.show(); return app.exec(); }</pre>

We would like to add a 'second' hand to the clock.
Which method does this change impact?

15. You need to add an auto completion facility to QT that provides completions to words entered into a line textbox or combobox widget. The completions are based on an item model. See the figure below for an example of the completer in action. You decide that the best way to do this is to create a new class *QCompleter* that provides this functionality.

qt-W1
qt-W2

Which widget classes need to change (i.e., add methods or variables) if *QCompleter* is added to QT?

Which class should *QCompleter* inherit from?

16. A view may be populated with data from any model such as a table or directory model. One such example of a view is shown below. Notice how the rows have an alternating background color to them. In order to implement this functionality, you need to add a method with the following signature.

qt-MV1
qt-MV2

```
void setAlternatingRowColors(bool enable)
```

Which class would you add this method to?

-
17. You are given the following bug description: qt-W1
Bug Title: Change foreground color of checkboxes inside tables qt-W2

Bug Description: When a checkbox is displayed in a table and the row is highlighted, the checkbox rectangle should be drawn with the same color as the text in the cell.

Which class (or classes) will most likely need to be changed in order to fix this bug?

-
18. You need to extend Qt's functionality to include an image plugin to read and write images using an new image format you invented. qt-I1
 qt-I2

Which class do you need to subclass for this plugin?

-
19. Syntax Highlighting is the use of appropriate fonts and colors to highlight different elements in a document. You need to create a new class *CodeHighlighter* that adds syntax highlighting to the *QTextEdit* widget. *QTextEdit* is used to edit and display both plain and rich text. qt-W1
 qt-W2

- a) Which abstract class would you derive *CodeHighlighter* from?
 b) Which method would *CodeHighlighter* need to reimplement from the abstract class?

- c) You will need to instantiate *CodeHighlighter* with *QTextEdit*'s underlying *QTextDocument* as the parent. Which method in *QTextEdit* returns a text document?

-
20. OpenGL is a standard API for rendering 3D graphics. To use OpenGL-enabled widgets in a Qt application, which class do you need to subclass? qt-G1
 qt-G2

-
21. You need to improve the design of the module shown in the following diagram and remove some redundancy. qt-I1
 qt-I2

Which method will most probably need to be defined in *QIODevice* from it's subclasses? (Note that the method can still

be reimplemented in *QIODevice* subclasses as needed.)

Should this field be private, public or protected?

22. The class <i>QHBoxLayout</i> provides horizontal geometry management for its child widgets. All the horizontal box's child widgets are placed alongside each other and sized accordingly. See the following example of <i>QHBoxLayout</i> in action.	qt-L1
	qt-L2

Now consider the following code fragments.

Code Fragment 1

```
QHBoxLayout *hbox = new QHBoxLayout;
QPushButton *child1 = new QPushButton(hbox);
QPushButton *child2 = new QPushButton(hbox);
```

Code Fragment 2

```
QWidget *hbox = new QWidget;
QPushButton *child1 = new QPushButton;
QPushButton *child2 = new QPushButton;

QHBoxLayout *layout = new QHBoxLayout;
layout->addWidget(child1);
layout->addWidget(child2);
hbox->setLayout(layout);
```

- a) True or False: Code Fragment 1 and Code Fragment 2 are equivalent. In other words, when executed, they produce the same result.
 - b) True or False: *QHBoxLayout* can be removed without causing any code to break. Base your answer on information given in the diagram.
 - c) True or False: *QHBoxLayout* can be removed without causing any code to break. Base your answer on information given in the diagram.
-

D.2. Comprehension Questions for the wxWidgets system

Questions on the wxWidgets System	Diagrams Shown ⁸
1. What is the class stereotype assigned to <i>wxDbTableInf</i> ?	wx-D1 wx -D2
2. Which classes are dependent on <i>wxRect</i> ?	wx-G1 wx-G2
3. Is this statement true or false? The boundary class <i>wxDC</i> is associated with the entity class <i>wxBitmap</i> via the attribute <i>m_selectedBitmap</i> .	wx-I1 wx-I2
4. Which class is involved in an aggregation relationship with <i>wxObject</i> and is also associated with <i>wxHashTable</i> ?	wx-E1 wx-E2
5. wxWidget's document/view architecture lets you model your application primarily in terms of documents, which store data and views, which display and manipulate the data. Which concrete class (or classes) implement the <i>document</i> component in the document/view architecture?	wx-DV1 wx-DV2
6. Splash screen is a term used to describe an image that appears while a computer program is loading. Splash screens sometimes do not cover the entire screen, but only a rectangle near the center. Select the class responsible for setting the image/bitmap for the splash screen.	wx-MW1 wx-MW2
7. Which abstract boundary class is responsible for managing layout for the graphical user interface?	wx-L1 wx-L2
8. Which class models the relationship between a document class and a view class?	wx-DV1 wx-DV2
9. SVG is an XML-based file format and language for describing 2 dimensional graphics. Which class acts as an input handler for the SVG file format?	wx-G1 wx-G2

⁸ One of the two diagrams were shown for each question. The first one is in multi-cluster layout, the second is in orthogonal layout

10. Which class provides a control that models hierarchical data?	wx-W1
	wx-W2

11. Which concrete layout related classes and widget related classes are used in the Dialog box below?	wx-L1
	wx-L2

Select all layout related concrete classes and widget related concrete classes that apply.

12. Which command event class is used to give an application the chance to update various user interface elements?	wx-E1
	wx-E2

13. Which class lets you view information from a database table in a grid?	wx-D1
	wx-D2

14. Consider the Regions wxWidgets application shown below. The orange filled squares in the first row and third column of the 'Regions' frame window need to be changed to blue.	wx-MW1
	wx-MW2

Which method does this change impact?

15. You need to add undo/redo functionality to commands which are actions usually performed by selecting a control such as a menu item, pressing a button or any other means provided by the application to change the data or view.	wx-W1
	wx-W2

Which classes need to change (i.e., add methods or attributes) if undo/redo is to be implemented?

16. You need to add a method to the document/view component that returns a reference to the list of documents. The signature of the method to be added is	wx-DV1
wx-DV2 |
|---|------------------|

wxList& GetDocuments()

Which class would you add this method to?

17. You are given the following bug description:	wx-W1
Bug Title: SetForegroundColour doesn't update the control.	wx-W2

Bug Description:

wxStaticText::SetForegroundColour does not update the control to change the colour. The following code does not work after the control has been created.

```
bool set_to_red;
wxStaticText *t;
t->SetForegroundColour(set_to_red ?
wxColour("RED") : wxColour("GREEN"));
```

Which class (or classes) will most likely need to be changed in order to fix this bug?

18. You need to extend wxWidget's functionality to include an image handler to read and write images using an new image format you invented.	wx-I1
wx-I2 |
|--|----------------|

Which class do you need to subclass to support this new format?

19. wxSpellChecker is a generic spell-check component that can use different spell checking engines. You need to add engine interface support for the GNU 'Aspell' engine. A new class called <i>AspellInterface</i> needs to be created.	wx-W1
wx-W2 |
|---|----------------|

a) Which class would you derive *AspellInterface* from?

b) Which method would need to be reimplemented in *AspellInterface* from the base class?

c) You also need to add a dialog interface to your spellchecker similar to the one shown below. A new class *MySpellingDialog* needs to be created. Which class would you derive *MySpellingDialog* from?

Example showing spellchecker in action

-
20. OpenGL is a standard API for rendering 3D graphics. Which class do you need to subclass to create an OpenGL enabled window?	wx-G1
wx-G2 |
|---|----------------|
-

21. You need to improve the design of the module shown in the following diagram and remove some redundancy.	wx-I1
wx-I2 |
|---|----------------|
-

Which attribute (field) will most probably need to be defined in *wxDC* from it's subclasses?

Should this field be private, public or protected?

-
23. Sizers are used to define layout of controls in dialogs.
<i>wxNotebookSizer</i> is a specialized sizer to make sizers work in connection with using notebooks. This sizer is different from any other sizer as you must not add any children to it - instead, it queries the notebook class itself. The only thing this sizer does is to determine the size of the biggest page of the notebook and report an adjusted minimal size. An example of a notebook control is shown below. | wx-L1
wx-L2 |
|--|----------------|
-

Now consider the following code fragments.

Code Fragment 1

```
f=wxFrame()
mainsizer=wxBoxSizer(wxVERTICAL)
f.SetSizer(mainsizer)
```

```
notebook=wxNotebook()
```

```
notesizer = wxNotebookSizer(notebook)
mainsizer.Add(notesizer)
```

Code Fragment 2

```
f=wxFrame()
mainsizer=wxBoxSizer(wxVERTICAL)
f.SetSizer(mainsizer)
```

```
notebook=wxNotebook()
notesizer = wxBoxSizer(notebook)
mainsizer.Add(notesizer)
```

- a) True or False: Code Fragment 1 and Code Fragment 2 are equivalent. In other words, when executed, they produce the same result.
 - b) True or False: *wxNotebookSizer* can be removed without causing any code to break. Base your answer on information given in the diagram.
 - c) True or False: *wxBoxSizer* can be removed without causing any code to break. Base your answer on information given in the diagram.
-

D.3. Preference Ratings Questionnaire

For preference ratings, the following template was used for each question for two of the questions in Qt and wxWidgets for the main study.

Comprehension Rating	Very Useful	Somewhat Useful	Don't know	Somewhat Useless	Completely Useless
<system>-<module>1					
<system>-<module>2					
<system>-<module>3					

Aesthetic Rating	Very Appealing	Somewhat Appealing	Don't know	Somewhat Unappealing	Very Unappealing
<system>-<module>1					
<system>-<module>2					
<system>-<module>3					

Comments on your comprehension and aesthetic rating:

D.4. Qt Diagrams

The last number of the diagram ID on the lower right corner denotes the type of layout:

1= multi cluster layout, 2=orthogonal layout

qt-D1

qt-I1

D.5. wxWidgets Diagrams

The last number of the diagram ID on the lower right corner denotes the type of layout:

1= multi cluster layout, 2=orthogonal layout

wx-D1

wx-I2

wx-L1

wx-L2

wx-MW1

wx-W1

APPENDIX E

Study Material for Chapter 7

The diagrams are grouped by system. Each diagram within each system has the pattern name located at the lower right hand corner of the diagram.

E.1. JHotDraw Diagrams in Multi-cluster Layout

JHotDraw-Composite1

JHotDraw-Observer1

JHotDraw-Singleton1

JHotDraw-Strategy1

E.2. JHotDraw Diagrams in Orthogonal Layout

JHotDraw-Composite2

JHotDraw-Observer2

JHotDraw-Singleton2

JHotDraw-Singleton2

E.3. JUnit Diagrams in Multi-cluster Layout

JUnit-Composite1

JUnit-Composite1

JUnit-Observer1

E.5. Qt Diagrams in Multi-cluster Layout

qt-Strategy1

qt-Strategy1

qt-Singleton1

qt-Singleton2

qt-Singleton2

APPENDIX F

Study Material for Chapter 8

The diagrams are grouped by design pattern. Each diagram within each system has the pattern name located at the lower right hand corner of the diagram. The number 1 indicates multi-cluster, 2 indicates orthogonal layout.

F.1. Composite Pattern Diagrams

Which three classes participate in the Composite design pattern?
Name the role (*Composite*, *Component*, *Leaf*) of each class.

JHotDraw-Composite1

Which two classes participate in the Observer design pattern? Name the role (*Subject*, *Observer*) of each class.

JUnit-Observer2

F.3. Strategy Pattern Diagrams

Which three classes participate in the Strategy design pattern?
Name the role (Strategy, Concrete Strategy, Context) of each class.

qt-Strategy1

qt-Strategy1

**Which three classes participate in the Strategy design pattern?
Name the role (*Strategy*, *Concrete Strategy*, *Context*) of each class.**

JHotDraw-Strategy2

F.4. Singleton Pattern Diagrams

Name two Singleton classes. State briefly why you think they are singletons.

qt-Singleton1

Name two Singleton classes. State briefly why you think they are singletons.

JHotDraw-Singleton2

JHotDraw-Singleton2

REFERENCES

- [Alphonse, Caspersen, Decker 2007] Alphonse, C., Caspersen, M., and Decker, A., (2007), "Killer "killer examples" for Design Patterns", in Proceedings of 38th SIGCSE Technical Symposium on Computer Science Education, Covington, Kentucky, Mar. 7-10, pp. 228 - 232.
- [Ambler 2002] Ambler, S. W.,(2002),*The Elements of UML Style*, New York, NY, Cambridge University Press.
- [Andriyevska, Dragan, Simoes, Maletic 2005] Andriyevska, O., Dragan, N., Simoes, B., and Maletic, J. I., (2005), "Evaluating UML Class Diagram Layout based on Architectural Importance", in Proceedings of 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT), Budapest, Hungary, September 25th, pp. 14-19.
- [Arisholm, Briand, Hove, Labiche 2006] Arisholm, E., Briand, L. C., Hove, S. E., and Labiche, Y., (2006), "The Impact of UML Documentation on Software Maintenance: An Experimental Evaluation", *IEEE Transactions on Software Engineering (TSE)*, vol. 32, no. 6, pp. 365-381.
- [Aschwanden, Crosby 2006] Aschwanden, C. and Crosby, M. E., (2006), "Code Scanning Patterns in Program Comprehension", in Proceedings of 39th Hawaii International Conference on System Sciences (HICSS), Kauai, Hawaii.
- [Atkinson, Kuhne, Henderson-Sellers 2002] Atkinson, C., Kuhne, T., and Henderson-Sellers, B., (2002), "Stereotypical Encounters of the Third Kind", in Proceedings of UML, pp. 100-114.

- [Battista 1997] Battista, D. G., (1997), "An Experimental Comparison of Four Graph Drawing Algorithms", in Proceedings of 11 th Annual ACM Symposium on Computational Geometry: Theory and Applications, Vancouver, CA, April, pp. 303 - 325
- [Battista, Eades, Tamassia, Ioannis 1999] Battista, D. G., Eades, P., Tamassia, R., and Ioannis, T. G.,(1999),*Graph Drawing: Algorithms for the visualization of graphs*.
- [Battista, Eades, Tamassia, Tollis 1994] Battista, D. G., Eades, P., Tamassia, R., and Tollis, I., (1994), "Algorithms for drawing graphs: An annotated bibliography", *Computational Geometry Theory and Applications*, vol. 4, no. 5, pp. 235-282.
- [Bednarik, Myller, Sutinen, Tukiainen 2005] Bednarik, R., Myller, N., Sutinen, E., and Tukiainen, M., (2005), "Effects of Experience on Gaze Behavior during Program Animation", in Proceedings of 17th Annual Psychology of Programming Interest Group Workshop (PPIG'05), Brighton, UK, June 28 - July 1, pp. 49-61.
- [Bednarik, Tukiainen 2004] Bednarik, R. and Tukiainen, M., (2004), "Visual Attention Tracking during Program Debugging", in Proceedings of Third Nordic Conference on Human-Computer Interaction (NordiCHI'04), Tampere, Finland, pp. 331 - 334
- [Bednarik, Tukiainen 2005] Bednarik, R. and Tukiainen, M., (2005), "Effects of Display Blurring on the Behavior of Novices and Experts during Program Debugging", in Proceedings of Conference on Human Factors in Computing Systems (CHI), Portland, OR, USA, April 2-7, pp. 1204 - 1207.
- [Bednarik, Tukiainen 2006] Bednarik, R. and Tukiainen, M., (2006), "An Eye-tracking Methodology for Characterizing Program Comprehension Processes", in Proceedings of Symposium on Eye tracking research & Applications (ETRA), San Diego, California, pp. 125-132.

- [Bednarik, Tukiainen 2007] Bednarik, R. and Tukiainen, M., (2007), "Validating the Restricted Focus Viewer: A Study Using Eye-movement Tracking", *Behavior Research Methods*, vol. 39, no. 2, pp. 274-282.
- [Bednarik, Tukiainen 2008] Bednarik, R. and Tukiainen, M., (2008), "Temporal Eye-tracking Data: Evolution of Debugging Strategies with Multiple Representations", in Proceedings of Symposium on Eye Tracking Research & Applications (ETRA), Savannah, Georgia, pp. 99-102.
- [Beymer, Russell 2005] Beymer, D. and Russell, D. M., (2005), "WebGazeAnalyzer: a system for capturing and analyzing web reading behavior using eye gaze", in Proceedings of CHI '05 extended abstracts on Human factors in computing systems, Portland, OR, USA, pp. 1913-1916.
- [Binkley, Davis, Lawrie, Morrell 2009] Binkley, D., Davis, M., Lawrie, D., and Morrell, C., (2009), "To CamelCase or Under_score", in Proceedings of 17th IEEE International Conference on Program Comprehension (ICPC), Vancouver, Canada, pp. 158-167.
- [Binkley, Lawrie, Maex, Morrell 2009] Binkley, D., Lawrie, D., Maex, S., and Morrell, C., (2009), "Identifier Length and Limited Programmer Memory", *Science of Computer Programming*, vol. 74, pp. 430-445.
- [Bojko 2005] Bojko, A., (2005), "Eye Tracking in User Experience Testing: How to Make the Most of It. " in Proceedings of 14th Annual Conference of the Usability Professionals Association (UPA), Montréal, Canada.
- [Booch 1999] Booch, G., (1999), "UML in Action", *Communications of the ACM*, vol. 42, no. 10, pp. 26-28.
- [Booch, Jacobson, Rumbaugh 1999] Booch, G., Jacobson, I., and Rumbaugh, J., (1999), *The Unified Software Development Process*, Addison-Wesley.

- [Booch, Rumbaugh, Jacobson 2005] Booch, G., Rumbaugh, J., and Jacobson, I., (2005), *Unified Modeling Language User Guide*, 2nd ed., NJ, Addison Wesley.
- [Bouman 1973] Bouman, H., (1973), "Visual Interference in the Parafoveal Recognition of Initial and Final Letters of Words", *Vision Research*, vol. 13, pp. 767-782.
- [Brooks 1983] Brooks, R., (1983), "Towards a Theory of the Comprehension of Computer Programs", *International Journal of Man-Machine Studies*, vol. 18, no. 6, pp. 543-554.
- [Butler, Wermelinger, Sharp 2009] Butler, S., Wermelinger, M., and Sharp, H., (2009), "Relating Identifier Naming Flaws and Code Quality: An Empirical Study", in Proceedings of 16th Working Conference on Reverse Engineering (WCRE), Lille, France, October, pp. 13-16.
- [Byelas, Bondarev, Telea 2006] Byelas, H., Bondarev, E., and Telea, A., (2006), "Visualization of Areas of Interest in Component-Based System Architectures", in Proceedings of 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA), Cavtat/Dubrovnik, Croatia, August 29 - September 1, pp. 160-169.
- [Byelas, Telea 2006] Byelas, H. and Telea, A., (2006), "Visualization of Areas of Interest in Software Architecture Diagrams", in Proceedings of Symposium on Software Visualization, Brighton, UK, pp. 105-114.
- [Byelas, Telea 2008a] Byelas, H. and Telea, A., (2008a), "The Metric Lens: Visualizing Metrics and Structure on Software Diagrams", in Proceedings of 15th Working Conference on Reverse Engineering (WCRE), Antwerp, Belgium, October 15-18, pp. 339-340.
- [Byelas, Telea 2008b] Byelas, H. and Telea, A., (2008b), "Texture-based Visualization of Metrics on Software Architectures", in Proceedings of Symposium on Software Visualization, Ammersee, Germany, September 16-17, pp. 205-206.

- [Byelas, Telea 2009a] Byelas, H. and Telea, A., (2009a), "Towards Realism in Drawing Areas of Interest On Architecture Diagrams", *Journal of Visual Languages and Computing (JVLC)*, vol. 20, no. 2, pp. 110-128.
- [Byelas, Telea 2009b] Byelas, H. and Telea, A., (2009b), "Visualizing Metrics on Areas of Interest in Software Architecture Diagrams", in *Proceedings of IEEE Pacific Visualization Symposium (PacificVis)*, Beijing, China, April 20-23, pp. 33-40.
- [Byelas, Telea 2009c] Byelas, H. and Telea, A., (2009c), "Visualizing Multivariate Attributes on Software Diagrams", in *Proceedings of 13th European Conference on Software Maintenance and Reengineering (CSMR)*, Kaiserslautern, Germany, March 24-27, pp. 335-338.
- [Caprile, Tonella 2000] Caprile, B. and Tonella, P., (2000), "Restructuring Program Identifier Names", *16th IEEE International Conference on Software Maintenance (ICSM)*, pp. 97-107.
- [Card, Mackinlay, Shneiderman 1999] Card, S. K., Mackinlay, J., and Shneiderman, B., (1999), *Readings in Information Visualization Using Vision to Think*, San Francisco, CA, Morgan Kaufmann.
- [Card et al. 2001] Card, S. K., Pirolli, P., van der Wege, M. M., Morrison, J. B., Reeder, R. W., Schraedley, P. K., and Boshart, J., (2001), "Information Scent as a Driver of Web Behavior Graphs: Results of a Protocol Analysis Method for Web Usability", in *Proceedings of CHI*, pp. 498-505.
- [Christensen 2004] Christensen, H. B., (2004), "Frameworks: putting design patterns into perspective", *9th SIGCSE Conference on Innovation and Technology in Computer Science Education*, pp. 142 - 145.

- [Coad, Lefebvre, De Luca 1999] Coad, P., Lefebvre, E., and De Luca, J.,(1999),*Java modeling in color with UML : enterprise components and process*, Prentice Hall.
- [Conallen 2003] Conallen, J.,(2003),*Building Web Applications with UML*, Boston, Addison-Wesley.
- [Cowen 2001] Cowen, L., (2001), *An Eye Movement Analysis of Web-Page Usability*, Lancaster University, UK.
- [Crosby, Stelovsky 1990] Crosby, M. E. and Stelovsky, J., (1990), "How do we read algorithms? A case study", *IEEE Computer*, vol. 23, no. 1, pp. 24-35.
- [Cutrell, Guan 2007] Cutrell, E. and Guan, Z., (2007), "What Are You Looking For? An Eye-tracking Study of Information Usage in Web Search", in *Proceedings of CHI*, San Jose, CA, USA, pp. 407-416.
- [de Kock, van Biljon, Pretorius 2009] de Kock, E., van Biljon, J., and Pretorius, M., (2009), "Usability Evaluation Methods: Mind the Gaps", in *Proceedings of Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists Vanderbijlpark*, Emfuleni, South Africa pp. 122-131.
- [Deißenböck, Pizka 2006] Deißenböck, F. and Pizka, M., (2006), "Concise and Consistent Naming", *Software Quality Journal*, vol. 14, no. 3, pp. 261-282.
- [Della, Clark 2000] Della, L. and Clark, D., (2000), "Teaching Object-Oriented Development with Emphasis on Pattern Application", in *Proceedings of Australasian conference on Computing education*, pp. 56-63.
- [Détienne 2002] Détienne, F., (2002), "Software design - cognitive aspects", F. Bott, Ed.: London ; New York : Springer.
- [Diaz, Petit, Serna 2002] Diaz, J., Petit, J., and Serna, M., (2002), "A Survey of Graph Layout Problems", *ACM Computing Surveys (CSUR)*, vol. 34, no. 3, Sept, pp. 313-356.

- [Duchowski 2003] Duchowski, A. T., (2003), *Eye Tracking Methodology: Theory and Practice*, London, Springer-Verlag.
- [Dukovich, Janzen 2009] Dukovich, A. and Janzen, D. S., (2009), "Design Patterns Go to Hollywood: Teaching Patterns with Multimedia", in Proceedings of 6th International Conference on Information Technology: New Generations, Las Vegas, Nevada, Apr 27-29, pp. 684-689.
- [Dwyer 2001] Dwyer, T., (2001), "Three dimensional UML using force directed layout", in Proceedings of Australian symposium on Information visualisation - Volume 9, Sydney, Australia, pp. 77-85.
- [Eichelberger 2001] Eichelberger, H., (2001), "SugiBib", in Proceedings of Graph Drawing, pp. 467-468.
- [Eichelberger 2002] Eichelberger, H., (2002), "Aesthetics of Class Diagrams", in Proceedings of 1st International Workshop on Visualizing Software for Understanding and Analysis (VISOFT), Washington, DC, USA, pp. 23 - 31.
- [Eichelberger 2003] Eichelberger, H., (2003), "Nice Class Diagrams Admit Good Design?" in Proceedings of ACM Symposium on Software Visualization (SoftVis), San Diego, USA, Jun 11-13, pp. 159-167.
- [Eichelberger 2005] Eichelberger, H., (2005), *Aesthetics and Automatic Layout of UML Class Diagrams*, Universitat Wuerzburg, Germany, Dissertation Thesis.
- [Eichelberger 2008] Eichelberger, H., (2008), "Automatic Layout of UML Use Case Diagrams", in Proceedings of 4th ACM Symposium on Software Visualization (SoftVis), Herrsching am Ammersee, Germany, September 16-17, pp. 105-114.
- [Eichelberger, Eldogan, Schmid 2009] Eichelberger, H., Eldogan, Y., and Schmid, K., (2009), "A Comprehensive Survey of UML Compliance in Current Modelling Tools", in *Lecture*

Notes in Informatics Software Engineering, P. Liggesmeyer, G. Engels, J. Münch, J. Dörr and N. Riegel, Eds., pp. 39-50.

[Eichelberger, Schmid 2009] Eichelberger, H. and Schmid, K., (2009), "Guidelines on the Aesthetic Quality of UML Class Diagrams", *Information and Software Technology*, vol. 51, no. 12, pp. 1686-1698.

[Eichelberger, von Gudenberg 2002] Eichelberger, H. and von Gudenberg, J. W., (2002), "On the Visualization of Java Programs", in *Proceedings of Lecture Notes In Computer Science*, pp. 295 - 306.

[Eichelberger, von Gudenberg 2003a] Eichelberger, H. and von Gudenberg, J. W., (2003a), "Demonstration of the Advanced Layout of UML Class Diagrams by SugiBib", in *Proceedings of International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)*.

[Eichelberger, von Gudenberg 2003b] Eichelberger, H. and von Gudenberg, J. W., (2003b), "UML Class Diagrams - State of the Art in Layout Techniques", in *Proceedings of International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)*, Amsterdam, pp. 30-34.

[Eiglsperger, Eppinger, Kaufmann 2003] Eiglsperger, M., Eppinger, F., and Kaufmann, M., (2003), "An Approach for Mixed Upward Planarization", *Journal of Graph Algorithms and Applications*, vol. 7, no. 2, pp. 203-220.

[Eiglsperger et al. 2004] Eiglsperger, M., Gutwenger, C., Kaufmann, M., Kupke, J., Jünger, M., Leipert, S., Klein, K., Mutzel, P., and Siebenhaller, M., (2004), "Automatic Layout of UML Class Diagrams in Orthogonal Style", in *Proceedings of Information Visualization*, pp. 189-208.

- [Eiglsperger, Kaufmann, Siebenhaller 2003] Eiglsperger, M., Kaufmann, M., and Siebenhaller, M., (2003), "A Topology-Shape-Metrics Approach for the Automatic Layout of UML Class Diagram", in Proceedings of SoftVis, San Diego, CA, USA, pp. 189-198.
- [Ellson et al. 2001] Ellson, J., Gansner, E., Koutsofios, L., North, C. S., and Woodhull, G., (2001), "Graphviz - Open Source Graph Drawing Tools", in Proceedings, pp. 483 - 484.
- [Epelboim et al. 1997] Epelboim, J., Booth, J. R., Ashkenazy, R., Arash, T., and Steinman, R. M., (1997), "Fillers and Spaces in Text: The Importance of Word Recognition During Reading", *Vision Research*, vol. 37, no. 20, pp. 2899-2914.
- [Evitts 2000] Evitts, P., (2000), *A UML Pattern Language*, Sams.
- [Fronk 2006] Fronk, A., (2006), "3D Visualisation of Code Structures in Java Software Systems", *SOFTVis*, pp. 145-146.
- [Furnas 1986] Furnas, G. W., (1986), "Generalized Fisheye Views", in Proceedings of Human Factors in Computing Systems, Boston, MA, pp. 16-23.
- [Gamma, Helm, Johnson, Vlissides 1995] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., (1995), *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley.
- [Gogolla, Henderson-Sellers 2002] Gogolla, M. and Henderson-Sellers, B., (2002), "Analysis of UML Stereotypes within the UML Metamodel", in Proceedings of UML, pp. 84-99.
- [Goldberg, Kotval 1999] Goldberg, J. H. and Kotval, X. P., (1999), "Computer Interface Evaluation Using Eye Movements: Methods and Constructs", *International Journal of Industrial Ergonomics*, vol. 24, pp. 631-645.
- [Goldberg et al. 2002] Goldberg, J. H., Stimson, M. J., Lewenstein, M., Scott, N., and Wichansky, A. M., (2002), "Eye tracking in web search tasks: design implications", in

- Proceedings of 2002 symposium on Eye tracking research & applications (ETRA), New Orleans, Louisiana, pp. 51-58.
- [Gómez-Martin, Jiménez-Díaz, Arroyo 2009] Gómez-Martin, M., Jiménez-Díaz, G., and Arroyo, J., (2009), "Teaching Design Patterns Using a Family of Games", in Proceedings of 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education, pp. 268-272.
- [Guéhéneuc 2004] Guéhéneuc, Y.-G., (2004), "A Reverse Engineering Tool for Precise Class Diagrams", in Proceedings of Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research, Markham, Canada, Oct 4-7, pp. 28-41.
- [Guéhéneuc 2006] Guéhéneuc, Y.-G., (2006), "TAUPE: towards understanding program comprehension", in Proceedings of 16th IBM Centers for Advanced Studies on Collaborative research (CASCON), Canada, Oct, pp. 1-13.
- [Guéhéneuc, Albin-Amiot 2004] Guéhéneuc, Y.-G. and Albin-Amiot, H., (2004), "Recovering Binary Class Relationships: Putting Icing on the UML Cake", in Proceedings of 19th annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'04), Vancouver, Canada, Oct 24-28, pp. 301-314.
- [Gutwenger et al. 2003a] Gutwenger, C., Jünger, M., Klau, G. W., Klein, K., Kupke, J., Leipert, S., and Mutzel, P., (2003a), "GoVisual: A Diagramming Software for UML Class Diagrams", in *Graph Drawing Software*, M. a. M. Jünger, P., Ed.: Springer-Verlag, pp. 257-278.
- [Gutwenger et al. 2001] Gutwenger, C., Jünger, M., Klein, K., Kupke, J., Leipert, S., and Mutzel, P., (2001), "Caesar Automatic Layout of UML Class Diagrams", in Proceedings of Graph Drawing, pp. 461-462.

- [Gutwenger et al. 2003b] Gutwenger, C., Jünger, M., Klein, K., Kupke, J., Leipert, S., and Mutzel, P., (2003b), "A New Approach for Visualizing UML Class Diagrams", in Proceedings of ACM Symposium on Software Visualization (SoftVis), pp. 179-188.
- [Hamer 2004] Hamer, J., (2004), "An Approach to Teaching Design Patterns using Musical Composition", in Proceedings of 9th annual SIGCSE conference on Innovation and technology in computer science education, Leeds, UK, pp. 156 - 160
- [Hammouda, Guldogan, Koskimies, Systa 2004] Hammouda, I., Guldogan, O., Koskimies, K., and Systa, T., (2004), "Tool-Supported Customization of UML Class Diagrams for Learning Complex System Models", in Proceedings of 12th IEEE International Workshop on Program Comprehension (IWPC), Italy, 2004, pp. 24.
- [Harel, Koren 2002] Harel, D. and Koren, Y., (2002), "Drawing Graphs with non-uniform Vertices", in Proceedings of Working Conference on Advanced Visual Interfaces (AVI'02), pp. 157-166.
- [Herman, Melancon, Marshall 2000] Herman, I., Melancon, G., and Marshall, S. M., (2000), "Graph Visualization and Navigation in Information Visualization: A Survey", *IEEE Transactions on Visualization and Computer Graphics*, vol. 6, no. 1, pp. 24-43.
- [Hornof, Halverson 2003] Hornof, A. and Halverson, T., (2003), "Cognitive Strategies and Eye Movements for Searching Hierarchical Computer Displays", in Proceedings of CHI, Fort Lauderdale, Florida, USA, April 5-10.
- [Huang, Sajeew, Lai 2006] Huang, X., Sajeew, A. S. M., and Lai, W., (2006), "A Scalable Algorithm for Adjusting Node-Node Overlaps", in Proceedings of International Conference on Computer Graphics, Imaging and Visualisation (CGIV'06), pp. 43 - 48.
- [Iqbal, Adamczyk, Zheng, Bailey 2005] Iqbal, S. T., Adamczyk, P. D., Zheng, X. S., and Bailey, B. P., (2005), "Towards an index of opportunity: understanding changes in mental

- workload during task execution", in Proceedings of SIGCHI conference on Human factors in computing systems, Portland, Oregon, USA, pp. 311-320.
- [Iqbal, Zheng, Bailey 2004] Iqbal, S. T., Zheng, X. S., and Bailey, B. P., (2004), "Task-evoked pupillary response to mental workload in human-computer interaction", in Proceedings of CHI '04 extended abstracts on Human factors in computing systems, Vienna, Austria, pp. 1477-1480.
- [Irwin, Churcher 2003] Irwin, W. and Churcher, N., (2003), "Object Oriented Metrics: Precision Tools and Configurable Visualizations", in Proceedings of 9th International Software Metrics Symposium (METRICS'03), pp. 112 - 123.
- [Jacob 1990] Jacob, R. J. K., (1990), "What you look at is what you get: eye movement-based interaction techniques", in Proceedings of SIGCHI conference on Human factors in computing systems: Empowering people, Seattle, Washington, United States, pp. 11-18.
- [Jacob 1995] Jacob, R. J. K., (1995), "Eye Tracking in Advanced Interface Design ", in *Virtual Environments and Advanced Interface Design*, New York W. Barfield, and T. A. Furness (eds.) , Oxford University Press, pp. 258-288.
- [Jansen, Blackwell, Marriott 2003] Jansen, A. R., Blackwell, A. F., and Marriott, K., (2003), "Restricted Focus Viewer: A Tool for Tracking Visual Attention", in *First International Conference on Theory and Application of Diagrams, (Diagrams 2000)*. Edinburgh, Scotland, UK: Springer Berlin / Heidelberg.
- [Jeanmart, Guéhéneuc, Sahraoui, Habra 2009] Jeanmart, S., Guéhéneuc, Y.-G., Sahraoui, H., and Habra, N., (2009), "Impact of the Visitor Pattern on Program Comprehension and Maintenance", in Proceedings of 3rd International Symposium on Empirical Software Engineering and Measurement, Lake Buena Vista, Florida, Oct 15-16, pp. 69-78.

- [Juristo, Moreno 2001] Juristo, N. and Moreno, A.,(2001),*Basics of Software Engineering Experimentation*, Englewood Cliffs, NJ, Kluwer Academic Publishers.
- [Just, Carpenter 1980] Just, M. and Carpenter, P., (1980), "A Theory of Reading: From Eye Fixations to Comprehension", *Psychological Review*, vol. 87, pp. 329-354.
- [Kagdi, Maletic 2007] Kagdi, H. and Maletic, J. I., (2007), "Onion Graphs for Focus+Context Views of UML Class Diagrams", in Proceedings of IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT'07), Banff, Canada, June 25-26, pp. 145-154.
- [Keschenau 2004] Keschenau, M., (2004), "Reverse Engineering of UML Specifications from Java Programs", in Proceedings of Companion to the 19th Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications, Vancouver, Canada, Oct 24-28, pp. 326-327.
- [Khiaat, Matsumoto, Ogasawara 2004] Khiaat, A., Matsumoto, Y., and Ogasawara, T., (2004), "Task Specific Eye Movements Understanding for a Gaze-Sensitive Dictionary", in Proceedings of Conference on Intelligent User Interfaces, Funchal, Madeira, Portugal, pp. 265-267.
- [Kollman, Gogolla 2002] Kollman, R. and Gogolla, M., (2002), "Metric-Based Selective Representation of UML Diagrams", in Proceedings of 6th European Conference on Software Maintenance and Reengineering (CSMR'02), Budapest, Hungary, March 11-13, pp. 89 - 98.
- [Kollman, Selonon, Stroulia, Zündorf 2002] Kollman, R., Selonon, P., Stroulia, E., and Zündorf, A., (2002), "A Study in the Current State of the Art in Tool-supported UML-based Static Reverse Engineering", in Proceedings of Ninth Working Conference on Reverse Engineering (WCRE'02), Richmond, Virginia, Oct 29-Nov 1, pp. 22-34.

- [Koning, Dormann, van Vliet 2002] Koning, H., Dormann, C., and van Vliet, H., (2002), "Practical Guidelines for the Readability of IT-Architecture Diagrams", in Proceedings of 20th Annual International Conference on Systems Documentation (SIGDOC'02), Toronto, Canada, October 20-23.
- [Köth, Minas 2001] Köth, O. and Minas, M., (2001), "Abstraction in Graph-Transformation Based Diagram Editors", *Electronic Notes in Theoretical Computer Science*, vol. 50, no. 3.
- [Kunz 2006] Kunz, P. F., (2006), "HippoDraw Users Guide", Date Accessed: 03/05/2009, <http://www.slac.stanford.edu/grp/ek/hippodraw/>.
- [Kuzniarz, Staron, Wohlin 2004] Kuzniarz, L., Staron, M., and Wohlin, C., (2004), "An Empirical Study on Using Stereotypes to Improve Understanding of UML Models", in Proceedings of 12th International Workshop on Program Comprehension (IWPC) pp. 14-23.
- [Lai, Eades 2002] Lai, W. and Eades, P., (2002), "Removing edge-node intersections in drawings of graphs", *Information Processing Letters*, vol. 81, pp. 105-110.
- [Lange, Chaudron 2007] Lange, C. F. J. and Chaudron, M. R. V., (2007), "Interactive Views to Improve the Comprehension of UML Models - An Experimental Validation", *International Conference on Program Comprehension (ICPC'07)*, pp. 221-230.
- [Lange, Wijns, Chaudron 2007] Lange, C. F. J., Wijns, M. A. M., and Chaudron, M. R. V., (2007), "MetricViewEvolution: UML-based Views for Monitoring Model Evolution and Quality", in Proceedings of 11th European Conference on Software Maintenance and Reengineering (CSMR), pp. 327-328.

- [Lanza, Ducasse 2001] Lanza, M. and Ducasse, S., (2001), "A Categorization of Classes based on the Visualization of their Internal Structure: the Class Blueprint", in Proceedings of OOPSLA, pp. 300-311.
- [Lawrie, Morrell, Feild, Binkley 2006] Lawrie, D., Morrell, C., Feild, H., and Binkley, D., (2006), "What's in a Name? A Study of Identifiers", *14th International Conference on Program Comprehension (ICPC)*, pp. 3-12.
- [Littman, Pinto, Letovsky, Soloway 1986] Littman, D. C., Pinto, J., Letovsky, S., and Soloway, E., (1986), "Mental models and Software Maintenance", in Proceedings of Papers presented at the first workshop on empirical studies of programmers on Empirical studies of programmers, pp. 80-98.
- [MagicDraw] MagicDraw, "MagicDraw", Date Accessed: 3/13/2009, <http://www.magicdraw.com/>.
- [Matsuda, Uwano, Ohira, Matsumoto 2009] Matsuda, Y., Uwano, H., Ohira, M., and Matsumoto, K.-i., (2009), "An Analysis of Eye Movements during Browsing Multiple Search Results Pages ", in *Lecture Notes in Computer Science*, Springer Berlin, pp. 121-130.
- [Matzko et al. 2002] Matzko, S., Clarke, P. J., Gibbs, T. H., Malloy, B. A., Power, J. F., and Monahan, R., (2002), "Reveal: A Tool to Reverse Engineer Class Diagrams", in Proceedings of 40th International Conference on Tools Pacific: Objects for Internet, Mobile and Embedded Applications, Sydney, Australia, 2002, pp. 13-21.
- [Miller 1956] Miller, G. A., (1956), "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information", *The Psychological Review*, vol. 63, pp. 81-97.

- [Misue, Eades, Lai, Sugiyama 1995] Misue, K., Eades, P., Lai, W., and Sugiyama, K., (1995), "Layout Adjustment and the Mental Map", *Journal of Visual Languages and Computing*, vol. 6, no. 2, pp. 183-210.
- [Murata 2006] Murata, A., (2006), "Eye-Gaze Input Versus Mouse: Cursor Control as a Function of Age", *International Journal of Human-Computer Interaction*, vol. 21, no. 1, pp. 1-14.
- [Musial, Jacobs 2003] Musial, B. and Jacobs, T., (2003), "Application of focus + context to UML ", in *Proceedings of the Australian symposium on Information visualisation - Volume 24* Adelaide, Australia Australian Computer Society, Inc., pp. 75-80
- [Nakamichi, Shima, Sakai, Matsumoto 2006] Nakamichi, N., Shima, K., Sakai, M., and Matsumoto, K.-i., (2006), "Detecting Low Usability Web Pages using Quantitative Data of Users' Behavior", in *Proceedings of 28th International Conference on Software Engineering (ICSE'06)*, Shanghai, China, pp. 569-576.
- [Nevalainen, Sajaniemi 2005] Nevalainen, S. and Sajaniemi, J., (2005), "Short-Term Effects of Graphical versus Textual Visualisation of Variables on Program Perception", *17th Annual Psychology of Programming Interest Group Workshop (PPIG'05)*, pp. 77-91.
- [Nevalainen, Sajaniemi 2006] Nevalainen, S. and Sajaniemi, J., (2006), "An Experiment on ShortTerm Effects of Animated versus Static Visualization of Operations on Program Perception", in *Proceedings of International Computing Education Research Workshop* , Sept 9-10, pp. 7-16.
- [Nielsen, Levy 1994] Nielsen, J. and Levy, J., (1994), "Measuring Usability, Preference vs. Performance", *Communications of the ACM*, vol. 37, no. 4, April, pp. 66-75.
- [NS] NS, "NSurvey", Date Accessed: 8/1/2007, <http://sourceforge.net/projects/nsurvey/>.
- [Oppenheim 1992] Oppenheim, A. N.,(1992),*Questionnaire Design, Interviewing and Attitude Measurement*, Pinter, London.

- [Pan et al. 2004] Pan, B., Hembrooke, H., Gay, G., Granka, L., Feusner, M., and Newman, J., (2004), "Determinants of web page viewing behavior: An eye-tracking study", in Proceedings of Eye tracking research & applications symposium on Eye tracking research & applications, San Antonio, Texas.
- [Pietinen et al. 2008] Pietinen, S., Bednarik, R., Glotova, T., Tenhunen, V., and Tukiainen, M., (2008), "A Method to Study Visual Attention Aspects of Collaboration: Eye-tracking Pair Programmers Simultaneously", in Proceedings of Symposium on Eye Tracking Research & Applications (ETRA), Savannah, Georgia, pp. 39-42.
- [Pirolli, Card 1999] Pirolli, P. and Card, S. K., (1999), "Information Foraging", *Psychological Review*, vol. 106, no. 4, pp. 643-675.
- [Pirolli, Card, van der Wege 2003] Pirolli, P., Card, S. K., and van der Wege, M. M., (2003), "The Effects of Information Scent on Visual Search in the Hyperbolic Tree Browser", *ACM Transactions on Computer-Human Interaction (TOCHI'03)*, vol. 10, no. 1, Mar 2003, pp. 20-53.
- [Price, Baecker, Small 1993] Price, B. A., Baecker, R. M., and Small, I. S., (1993), "A Principled Taxonomy of Software Visualization", *Journal of Visual Languages and Computing*, no. 4, June, pp. 3.
- [Purchase 1997] Purchase, C. H., (1997), "Which Aesthetic Has the Greatest Effect on Human Understanding?" in Proceedings of Graph Drawing, pp. 284-290.
- [Purchase 1998a] Purchase, C. H., (1998a), "The Effects of Graph Layout", in Proceedings of Australasian Computer Human Interaction Conference, Adelaide, S. Australia, November 29 - December 12, pp. 80-86.

- [Purchase 1998b] Purchase, C. H., (1998b), "Performance of Layout Algorithms: Comprehension not Computation", *Journal of Visual Languages and Computing*, vol. 9, pp. 647-657.
- [Purchase 2000] Purchase, C. H., (2000), "Effective Information Visualization: A Study of Graph Drawing Aesthetics and Algorithms", *Interacting with Computers*, vol. 13, no. 2000, pp. 147-162.
- [Purchase 2002] Purchase, C. H., (2002), "Metrics for Graph Drawing Aesthetics", *Journal of Visual Languages and Computing*, vol. 13, pp. 501-516.
- [Purchase, Alder, Carrington 2000] Purchase, C. H., Alder, J.-A., and Carrington, D., (2000), "User Preference of Graph Layout Aesthetics: A UML Study", *Lecture Notes in Computer Science*, pp. 5-18.
- [Purchase, Alder, Carrington 2002] Purchase, C. H., Alder, J.-A., and Carrington, D., (2002), "Graph Layout Aesthetics in UML Diagrams: User Preferences", *Journal of Graph Algorithms and Applications*, vol. 6, no. 3, pp. 255-279.
- [Purchase, Carrington, Alder 2000] Purchase, C. H., Carrington, D., and Alder, J.-A., (2000), "Experimenting with Aesthetics-Based Graph Layout", in *Proceedings of Diagrams*, Edinburgh, Scotland, UK, pp. 498-501.
- [Purchase, Carrington, Alder 2002] Purchase, C. H., Carrington, D., and Alder, J.-A., (2002), "Empirical Evaluation of Aesthetics-based Graph Layout", *Empirical Software Engineering*, vol. 7, pp. 233-255.
- [Purchase, Cohen, James 1997] Purchase, C. H., Cohen, R. F., and James, M. I., (1997), "An Experimental Study of the Basis for Graph Drawing Algorithms", *ACM Journal of Experimental Algorithms*, vol. 2, no. 4, pp. Article No. 4.

- [Purchase et al. 2001] Purchase, C. H., Colpoys, L., McGill, M., Carrington, D., and Britton, C., (2001), "UML Class Diagram Syntax: An Empirical Study of Comprehension", in Proceedings of Australian Symposium on Information Visualisation, Sydney, Australia, pp. 113-120.
- [Purchase, Hoggan, Gorg 2006] Purchase, C. H., Hoggan, E., and Gorg, C., (2006), "How Important is the "Mental Map"? - An Empirical Investigation of a Dynamic Graph Layout Algorithm", in Proceedings of Graph Drawing, pp. 184-195.
- [Purchase, McGill, Colpoys, Carrington 2001] Purchase, C. H., McGill, M., Colpoys, L., and Carrington, D., (2001), "Graph Drawing Aesthetics and the Comprehension of UML Class Diagrams: An Empirical Study", in Proceedings of Australian Symposium on Information Visualisation, Sydney, Australia, December, pp. 129-137.
- [Rayner 1998] Rayner, K., (1998), "Eye Movements in Reading and Information Processing: 20 Years of Research", *Psychological Bulletin*, vol. 124, no. 3, pp. 372-422.
- [Rayner, Fischer, Pollatsek 1998] Rayner, K., Fischer, M. H., and Pollatsek, A., (1998), "Unspaced Text Interferes with Both Word Identification and Eye Movement Control", *Vision Research*, vol. 38, no. 8, pp. 1129-1144.
- [Reeder, Pirolli, Card 2001] Reeder, R. W., Pirolli, P., and Card, S. K., (2001), *WebEyeMapper and WebLogger: tools for analyzing eye tracking data collected in web-use studies*, Seattle, Washington, ACM Press.
- [Reiss 2001] Reiss, S. P., (2001), "An Overview of BLOOM", *Workshop on Program Analysis for Software Tools and Engineering (PASTE)*, pp. 2-5.
- [Ricca et al. 2006] Ricca, F., Di Penta, M., Torchiano, M., Tonella, P., and Ceccato, M., (2006), "An empirical study on the usefulness of Conallen's stereotypes in Web application

- comprehension", in Proceedings of 8th IEEE International Symposium on Web Site Evolution (WSE), Philadelphia, PA, September 23-24, pp. 58-65.
- [Ricca et al. 2007] Ricca, F., Di Penta, M., Torchiano, M., Tonella, P., and Ceccato, M., (2007), "The Role of Experience and Ability in Comprehension Tasks supported by UML Stereotypes", in Proceedings of 29th International Conference on Software Engineering (ICSE), Minneapolis, May 19-27, pp. 375-384.
- [Ricca et al. 2010] Ricca, F., Di Penta, M., Torchiano, M., Tonella, P., and Ceccato, M., (2010), "How Developers' Experience and Ability Influence Web Application Comprehension Tasks Supported by UML Stereotypes: A Series of Four Experiments", *IEEE Transactions on Software Engineering*, vol. 36, no. 1, January/February, pp. 96-118.
- [Romero, Cox, du Boulay, Lutz 2002] Romero, P., Cox, R., du Boulay, B., and Lutz, R., (2002), "Visual Attention and Representation Switching During Java Program Debugging: A Study Using the Restricted Focus Viewer", *Diagrams 2002*, pp. 221-235.
- [Romero, Du Boulay, Lutz, Cox 2003] Romero, P., Du Boulay, B., Lutz, R., and Cox, R., (2003), "The Effects of Graphical and Textual Visualisations in Multi-Representational Debugging Environments", in Proceedings of IEEE Symposium on Human Centric Computing Languages and Environments, Washington, D.C., USA, pp. 236-238.
- [Ryall, Marks, Shieber 1997] Ryall, K., Marks, J., and Shieber, S., (1997), "An Interactive Constraint-Based System for Drawing Graphs", in Proceedings of ACM Symposium on User Interface Software and Technology, Banff, Alberta, Canada, pp. 97 - 104.
- [Sawant, Bali 2007] Sawant, A. P. and Bali, N., (2007), "DiffArchViz: A Tool to Visualize Correspondence Between Multiple Representations of a Software Architecture", in Proceedings of 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT'07), Banff Centre, Alberta, Canada, June 25-26.

- [Seemann 1997] Seemann, J., (1997), "Extending the Sugiyama Algorithm for Drawing UML Class Diagrams: Towards Automatic Layout of Object-Oriented Software Diagrams", in Proceedings of Proceedings of the 5th International Symposium on Graph Drawing, pp. 415 - 424.
- [SF] SF, "Sourceforge.net", Date Accessed: 8/1/2007, <http://sourceforge.net/>.
- [Sharif, Maletic 2009a] Sharif, B. and Maletic, J. I., (2009a), "The Effect of Layout on the Comprehension of UML Class Diagrams: A Controlled Experiment", in Proceedings of 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT), Edmonton, Canada, Sept 25, pp. 11-18.
- [Sharif, Maletic 2009b] Sharif, B. and Maletic, J. I., (2009b), "An Empirical Study on the Comprehension of Stereotyped UML Class Diagram Layouts", in Proceedings of 17th IEEE International Conference on Program Comprehension (ICPC), Vancouver, BC, Canada, May 17-19, pp. 268-272.
- [Sharif, Maletic 2010a] Sharif, B. and Maletic, J. I., (2010a), "The Effects of Layout on Detecting the Role of Design Patterns", in Proceedings of 23rd IEEE-CS International Conference on Software Engineering Education and Training (CSEE&T 2010), Carnegie Mellon University, Pittsburgh, USA, March 9-12, pp. 8 pages to appear.
- [Sharif, Maletic 2010b] Sharif, B. and Maletic, J. I., (2010b), "An Eye Tracking Study on CamelCase and under_score Identifier Styles", in Proceedings of IEEE International Conference on Program Comprehension (ICPC '10), Braga, Portugal, Jun 30-Jul 2, pp. 10 pages to appear.
- [Shumba 2005] Shumba, R., (2005), "Usability of Rational Rose and Visio in a Software Engineering Course", *The SIGCSE Bulletin*, vol. 37, no. 2, pp. 107-110.

- [Six, Tollis 2001] Six, J. M. and Tollis, I. G., (2001), "Effective Graph Visualization Via Node Grouping", *IEEE Symposium on Information Visualization (InfoVis)*, pp. 51-58.
- [Stacy, Macmillian 1995] Stacy, W. and Macmillian, J., (1995), "Cognitive Bias in Software Engineering", *Communications of the ACM*, vol. 38, no. 6, pp. 57-63.
- [Staron, Kuzniarz, Thurn 2005] Staron, M., Kuzniarz, L., and Thurn, C., (2005), "An Empirical Assessment Of Using Stereotypes to Improve Reading Techniques in Software Inspections", in *Proceedings of Proceedings of the third workshop on Software quality*, St. Louis, Missouri, pp. 1-7.
- [Staron, Kuzniarz, Wohlin 2006] Staron, M., Kuzniarz, L., and Wohlin, C., (2006), "Empirical assessment of using stereotypes to improve comprehension of UML models: a set of experiments", *Journal of Systems and Software*, vol. 79, pp. 727-742.
- [Storey 2005] Storey, M.-A. D., (2005), "Theories, Methods and Tools in Program Comprehension: Past Present and Future", *13th International Workshop on Program Comprehension*, May, pp. 181-191.
- [Storey, Fracchia, Müller 1997] Storey, M.-A. D., Fracchia, F. D., and Müller, H., (1997), "Cognitive Design Elements to Support the Construction of a Mental Model during Software Visualization", *International Workshop on Program Comprehension*, no. 17-28.
- [Storey, Müller 1995] Storey, M.-A. D. and Müller, H., (1995), "Graph Layout Adjustment Strategies", in *Proceedings of International Symposium on Graph Drawing (GD'95)*, Passau, Germany, pp. 487-499.
- [Storey, Wong, Fracchia, Müller 1997] Storey, M.-A. D., Wong, K., Fracchia, F. D., and Müller, H., (1997), "On Integrating Visualization Techniques for Effective Software Exploration", in *Proceedings of IEEE Symposium on Information Visualization (IV'97)*, Phoenix, AZ.

- [Sugiyama 1987] Sugiyama, K., (1987), "A Cognitive Approach to Graph Drawing", *Cybernetics and Systems*, vol. 18, pp. 447-488.
- [Sugiyama, Tagawa, Toda 1981] Sugiyama, K., Tagawa, S., and Toda, M., (1981), "Methods for Visual Understanding of Hierarchical System Structures", *IEEE Transactions on Systems, Man, and Cybernetics (SMC)*, vol. 11, no. 2, Feb, pp. 109-125.
- [Sun, Wong 2005] Sun, D. and Wong, K., (2005), "On Evaluating the Layout of UML Class Diagrams for Program Comprehension", in Proceedings of 13th IEEE International Workshop on Program Comprehension, St. Louis, Missouri, USA, pp. 317-328.
- [Sutter, Alexandrescu 2004] Sutter, H. and Alexandrescu, A., (2004), *C++ Coding Standards: 101 Rules, Guidelines, and Best Practices*, Addison-Wesley Professional.
- [Sutton, Maletic 2007] Sutton, A. and Maletic, J. I., (2007), "Recovering UML Class Models from C++: A Detailed Explanation", *Information and Software Technology*, vol. 49, no. 3, Jan, pp. 212-229.
- [Tilley, Huang 2003] Tilley, S. and Huang, S., (2003), "A qualitative assessment of the efficacy of UML diagrams as a form of graphical documentation in aiding program understanding", in Proceedings of 21st annual international conference on Documentation (SIGDOC), San Francisco, CA, USA, pp. 184-191.
- [Tobii 2001] Tobii, (2001), Tobii Technology, Date Accessed: 3/14/2010, <http://www.tobii.com/>.
- [Tonella, Potrich 2001] Tonella, P. and Potrich, A., (2001), "Reverse Engineering of the UML Class Diagram from C++ Code in the Presence of Weakly Typed Containers", in Proceedings of International Conference on Software Maintenance (ICSM'01), Florence, Italy, Nov 6-10, pp. 376-385.
- [Uwano, Monden, Matsumoto 2008] Uwano, H., Monden, A., and Matsumoto, K.-i., (2008), "DRESREM 2: An Analysis System for Multi-document Software Review Using

- Reviewers' Eye Movements", in Proceedings of 3rd International Conference on Software Engineering Advances (ICSEA), Sliema, Malta, pp. 177 - 183
- [Uwano, Nakamura, Monden, Matsumoto 2006] Uwano, H., Nakamura, M., Monden, A., and Matsumoto, K., (2006), "Analyzing individual performance of source code review using reviewers' eye movement", in Proceedings of 2006 symposium on Eye tracking research & applications (ETRA), San Diego, California, pp. 133-140.
- [von Gudenberg, Niederle, Ebner, Eichelberger 2006] von Gudenberg, J. W., Niederle, A., Ebner, M., and Eichelberger, H., (2006), "Evolutionary Layout of UML Class Diagrams", in Proceedings of Symposium on Software Visualization, Brighton, UK, Sept 4 - 5, pp. 163-164.
- [von Mayrhauser, Vans 1996] von Mayrhauser, A. and Vans, A. M., (1996), "Identification of dynamic comprehension processes during large scale maintenance", *IEEE Transactions on Software Engineering*, vol. 22, no. 6, pp. 424-437.
- [Vonder Embse 1987] Vonder Embse, C. B., (1987), *An eye fixation study of time factors comparing experts and novices while reading and interpreting mathematical graphs*, Ohio State University, Dissertation, UMI #8717747 Thesis.
- [VP 2007] VP, (2007), "Visual Paradigm for UML", Date Accessed: 1/14/2009, <http://www.visual-paradigm.com/product/vpuml/>.
- [Ware 2000] Ware, C.,(2000),*Information Visualization Perception for Design*, Morgan Kaufmann Publishers.
- [Ware, Purchase, Colpoys, McGill 2002] Ware, C., Purchase, H., Colpoys, L., and McGill, M., (2002), "Cognitive measurements of graph aesthetics", *Information Visualization*, vol. 1, no. 2, pp. 103-110.

- [Warren 2005] Warren, I., (2005), "Teaching Patterns and Software Design", in Proceedings of Australasian conference on Computing Education, Newcastle, Australia, pp. 39 - 49.
- [Weiss 2005] Weiss, S., (2005), "Teaching Design Patterns By Stealth", in Proceedings of 36th SIGCSE technical symposium on Computer science education, Missouri, pp. 492-494.
- [Whalen, Inkpen 2005] Whalen, T. and Inkpen, K. M., (2005), "Gathering evidence: use of visual security cues in web browsers", in Proceedings of 2005 conference on Graphics interface, Victoria, British Columbia, pp. 137-144.
- [Wiese, Eiglsperger, Kaufmann 2001] Wiese, R., Eiglsperger, M., and Kaufmann, M., (2001), "yFiles: Visualization and Automatic Layout of Graphs", in Proceedings of 9th International Symposium on Graph Drawing, pp. 453-454.
- [Wirfs-Brock 1994] Wirfs-Brock, R., (1994), "Adding to your Conceptual Toolkit: What's Important about Responsibility-Driven Design", *Report on Object Analysis and Design (ROAD)*, vol. 1, no. 2, July-Aug, pp. 27-34.
- [Wohlin et al. 2000] Wohlin, C., Runeson, P., Host, M., Ohlsson, M. C., Regnell, B., and Wesslen, A., (2000), *Experimentation in Software Engineering - An Introduction.*, Kluwer Academic Publishers.
- [Wong, Tilley, Muller, Storey 1995] Wong, K., Tilley, S., Muller, H., and Storey, M.-A. D., (1995), "Structural Redocumentation: A Case Study", *IEEE Software*, vol. 12, no. 1, pp. 46-54.
- [Yi, Wu, Gan 2004] Yi, T., Wu, F., and Gan, C., (2004), "A Comparison of Metrics for UML Class Diagrams", in Proc. of ACM SIGSOFT Software Engineering Notes, pp.1- 6.
- [Yusuf, Kagdi, Maletic 2007] Yusuf, S., Kagdi, H., and Maletic, J. I., (2007), "Assessing the Comprehension of UML Class Diagrams via Eye Tracking", in Proceedings of 15th IEEE Intl Conf on Program Comprehension (ICPC), Banff Canada, June 26-29, pp. 113-122.