

# Toward A Theory of Design Critiquing

- The Furniture Design Critic Program

Submitted in partial fulfillment of the requirements for the degree of

**DOCTOR OF PHILOSOPHY**

COMPUTATIONAL DESIGN

**CARNEGIE MELLON UNIVERSITY**

**Yeonjoo Oh**

Advisory Committees:

*Professor* **Mark D Gross** (Chair)  
School of Architecture  
Carnegie Mellon University

*Professor* **Ellen Yi-Luen Do**  
Colleges of Architecture and Computing  
Georgia Institute of Technology

*Professor* **Suguru Ishizaki**  
Department of English  
Carnegie Mellon University



## Acknowledgements

I am grateful to a number of people who have helped me conduct this research presented in this dissertation.

I am grateful to my dissertation committee for their intellectual support, sharp insights, and patience. Mark D Gross has always provided me with the good arguments and hard questions throughout my master and PhD student life. He also taught me how to write academic papers and programs as tools for articulating and developing my ideas. I appreciate his continuing and patient criticisms. He was always there to meet and talk about my ideas and to ask me good questions to help me think through my work. I am also indebted to Ellen Yi-Luen Do who has offered inspiring ideas and continuing encouragement since my academic life has started. Although she was in Atlanta for recent four years, I could not feel her absence, because she was always there to listen to my story and offer helpful comments. She also taught me how to live as a researcher and demonstrated how to work with great enthusiasm. The guidance and constructive feedback given by Suguru Ishizaki

was invaluable in developing the theoretical framework and the software, Furniture Design Critic. Without his sharp questions and insightful comments, I could not have finished this dissertation.

I also gained much from interactions with fellow graduate students at the CoDe Lab. I thank all my friends who offered their time for discussions over many years; Mike Weller, Gabe Johnson, Ken Camarata, and Eric Schweikardt. I also thank Soojin Jun, Sukbin Lee, Kwangjun Lee, Sanghoon Lee, Kuhn Park for their friendships and supports at CMU.

I thank my husband, Wanjei Cho for offering his thoughtful and continuing supports and all other things he has done for me and for listening to my frustrations. I could not have done this without his love and patience. I also thank my mother-in-law, Sosuk Park for her understanding and supports. I do not know how I could express all my appreciation to my parents, Yongmu Oh and Soyoung Lee, brother Youkeun Oh, sister-in-law Junghwa Chun and two adorable nieces, Yoonsuh and Hyunsuh. Without their supports and love, I could not follow my interests and finish my Ph.D work. Lastly, I would like to dedicate this dissertation to my parents.

# Table of Contents

Acknowledgements .....	3
List of Figures .....	11
List of Tables.....	15
Abstract .....	16
1. Introduction.....	18
1.1. Definitions of Terms.....	22
1.1.1. <i>Critiquing Methods and Conditions</i> .....	22
1.2. Motivations .....	23
1.2.1. <i>Lack of Understanding of Design Critiquing</i> .....	23
1.2.2. <i>Limitations of Computer-based Design Critiquing Systems</i> .....	24
1.3. Research Scope .....	25

1.4.	Approach and Contributions .....	27
1.5.	Guide to the Dissertation.....	28
2.	Computer-based Design Critiquing Systems .....	29
2.1.	Critiquing Process.....	31
2.2.	Rules.....	32
2.2.1.	<i>Forms of Rules</i> .....	32
2.2.2.	<i>Completeness of Knowledge (Comparative and Analytic Critiquing)</i> .....	35
2.2.3.	<i>Rule Management</i> .....	37
2.2.4.	<i>End User Rule Authoring</i> .....	41
2.3.	Intervention Techniques.....	42
2.3.1.	<i>Timing</i> .....	43
2.3.2.	<i>Activation</i> .....	44
2.3.3.	<i>Delivery Types</i> .....	45
2.3.4.	<i>Communication Modalities</i> .....	49
2.4.	Summary .....	51
3.	A Framework for Critiquing Practice in Design Studio .....	54
3.1.	A Process Model of Critiquing .....	55
3.2.	Fundamental Factors for Context-sensitive Critiquing.....	57
3.2.1.	<i>Critiquing Settings</i> .....	58
3.2.1.1.	Desk Crit(iques) .....	58
3.2.1.2.	Group Crit.....	59
3.2.1.3.	Interim Review.....	59
3.2.1.4.	Formal (Final) Review .....	60
3.2.1.5.	Informal Interaction.....	62

3.2.2. <i>Teacher-student Relationships</i> .....	63
3.2.2.1. Master and Apprentice .....	63
3.2.2.2. User and Designer .....	65
3.2.2.3. Peer Critiquing .....	65
3.2.3. <i>Communication Modalities</i> .....	66
3.2.3.1. Speech .....	66
3.2.3.2. Written Comments .....	67
3.2.3.3. Drawing: Graphic Annotation and Image .....	67
3.2.3.4. Gesture .....	67
3.2.3.5. Combining Modalities .....	68
3.2.4. <i>Delivery Types</i> .....	68
3.2.5. <i>Delivery</i> .....	69
3.2.6. <i>Critiquing Phases</i> .....	70
3.2.7. <i>Individual Differences</i> .....	71
3.2.7.1. Learning Style .....	71
3.2.7.2. Spatial Ability .....	72
3.2.7.3. Gender, Race, and Culture .....	73
3.2.8. <i>Students' Knowledge and Experience</i> .....	74
3.2.9. <i>Students' Response Types</i> .....	74
3.2.10. <i>Design Artifacts and Learning Goals</i> .....	75
3.3. <i>Summary: A Framework of Critiquing Practice</i> .....	75
4. <i>Intelligent Tutoring Systems</i> .....	80
4.1. <i>ITS System Architecture</i> .....	81
4.2. <i>Two Main Approaches of Intelligent Tutoring Systems</i> .....	83
4.2.1. <i>Model Tracing Tutors</i> .....	83

4.2.2. <i>Constraint-based Tutors</i> .....	85
4.2.3. <i>Constraint-based Tutors for Design Critiquing</i> .....	86
4.3. Summary .....	88
<b>5. The Furniture Design Critic</b> .....	<b>89</b>
5.1. The Program: Why a Computational Model?.....	89
5.2. The domain: Flat-pack Furniture Design .....	90
5.3. The “5x3” critiquing model: Delivery Types and Communication Modalities.....	93
5.4. An Example from Real Life: Analysis of an Desk Crit Session.....	95
5.5. Furniture Design Critic: Three Vignettes.....	99
5.5.1. <i>Vignette 1: Ann, a beginner</i> .....	103
5.5.2. <i>Vignette 2: Ben, a knowledgeable designer</i> .....	106
5.5.3. <i>Vignette 3: Claire</i> .....	112
5.5.4. <i>Summary</i> .....	116
<b>6. A Computational Model of Design Critiquing: How the Furniture Design Critic works</b> .....	<b>118</b>
6.1. A Closer Look at the Vignettes .....	118
6.2. Vignette 1: Ann.....	122
6.3. Vignette 2: Ben.....	126
6.4. Vignette 3: Claire.....	131
<b>7. Implementation Details</b> .....	<b>141</b>
7.1. Construction Interface .....	142

7.2.	Parser.....	143
7.3.	Design Constraints .....	146
7.4.	Pattern Matcher .....	149
7.5.	User Model.....	150
7.6.	Pedagogical Module .....	152
	7.6.1. <i>Selecting which feedback should be offered first</i> .....	154
	7.6.2. <i>Making an inference about a designer based on the User Model</i> .....	155
	7.6.3. <i>Selecting delivery types and communication modalities</i> .....	158
7.7.	Critiquing Rules.....	161
7.8.	Critic Presenter .....	161
7.9.	Critiquing Interface .....	163
7.10.	States .....	165
8.	Conclusions, Contributions, and Future Work.....	169
8.1.	Summary .....	169
8.2.	Contributions .....	170
8.3.	Future Work.....	172
	8.3.1. <i>Application in Studio</i> .....	172
	8.3.2. <i>Professional Development of Studio Teachers</i> .....	174
	8.3.3. <i>Expanding the Furniture Design Critic</i> .....	175
	8.3.4. <i>Test Effective Critiquing Methods</i> .....	176
	8.3.5. <i>Extension to Other Domains</i> .....	176
8.4.	Conclusion .....	177

<b>References .....</b>	<b>179</b>
-------------------------	------------

## List of Figures

Figure 2.1 Robbins' Critiquing Process (1998): Activate – Detect – Advise – Improve – Record .....	31
Figure 2.2 Fischer's Critiquing Process (1998) .....	32
Figure 2.3 Janus Family: (a) a user places kitchen appliances in the 2D working window and the system presents written feedback; (b) the system presents argumentation and positive or counter example kitchen layouts. ....	34
Figure 2.4 SEDAR: the system offers textual critiques in the bottom window, "masonry chimney1 must be at least 1 foot away from masonry chimney2" and highlights relevant design objects in color .....	38
Figure 2.5 Argo: (a) a user draws UML diagrams that represent software components and the interactions between them; (b) Argo presents feedback in the To Do List window. When a user clicks the name of the critique, the system offers a detailed explanation of the error. ....	39
Figure 2.6 Lisp Critic: the large window is the user's work environment. The Lisp-Critic window on top of the working window displays critiques. Here it presents a 'cond-to-if' transformation and explains why the system recommends changing the cond to the if expression. ....	40
Figure 2.7 Java Critiquer: this system supports programming teachers. A teacher can review, modify, add or remove system-generated critiques. The teacher can write new rules using regular expressions. ....	42

Figure 2.8 Solibri Model Checker: when a user selects an item (in this case, space ventilation) in the left window, the system presents its review, then the system highlights relevant parts in the 3D model. ....	44
Figure 2.9 CodeBroker: retrieves and offers library components such as classes and methods .....	46
Figure 2.10 Design Evaluator: the system offers feedback in several ways: text feedback, graphical annotations and 3D visualization (VRML model).....	47
Figure 2.11 Han’s Universal Code Checker: the system checks the design against building codes and reports the evaluation results on the web pages by showing simulations on VRML models and graphic annotations on CAD drawings. ....	48
Figure 2.12 DAISY: the user draws feature and class diagrams using UML. DAISY presents textual critiques in the “Things to take care of” window and displays a correct graphical notation beside the user’s diagram. ....	50
Figure 3.1 Critiquing Steps: (1) Observation; (2) Noticing; (3) Identification; (4) Sequence; (5) Delivery Types and Communication Modalities; and (6) Delivery .....	56
Figure 3.2 Three Perspectives of Critiquing Settings: (1) Numbers of Students; (2) Public – Private; and (3) Informal – Formal. ....	62
Figure 3.3 A Framework for Critiquing Practice: Conditions and Methods. A studio teacher considers critiquing conditions and then selects a set of critiquing methods to offer feedback.....	78
Figure 4.1 System Architecture of an Intelligent Tutoring System .....	81
Figure 5.1 Final design submissions of several students for their Flat-pack Furniture Making Exercises: (a) Gregory Zulkie; (b) David Rocco; (c) Mary Katica .....	91
Figure 5.2 Wooden stool and drawing including a plan, elevation, section, and an axonometric view (designer: Lee Byron).....	93
Figure 5.3 The Furniture Design Critic interface is composed of Sketch Interface and 3D Model Interface	101
Figure 5.4 The Furniture Design Critic observes Ann’s bookcase design. The Critic points out a problem and explains why it must be fixed. Also the Critic annotates Ann’s diagram. The red arrows indicate the vertical loads .....	104

Figure 5.5 The Furniture Design Critic points out a design problem with evaluative feedback, and annotates Ann's design. A red arrow indicates lateral loads on the bookcase. ....	105
Figure 5.6 Finding no further issue to point out, the Critic merely suggests more exploration.....	106
Figure 5.7 The Furniture Design Critic observes and interprets Ben's table design. The Critic also introduces an idea that Ben has perhaps not considered. ....	108
Figure 5.8 The Furniture Design Critic examines Ben's table design and describes what it sees. It introduces an idea that Ben may not think about and demonstrates how to resolve a problem – two unsupported corners. It also annotates Ben's diagram to help him understand the written comments. ....	109
Figure 5.9 The Furniture Design Critic thinks aloud about Ben's table design. It also demonstrates a brace by marking up Ben's diagram and writes an evaluative comment. ....	110
Figure 5.10 The Furniture Design Critic suggests more design exploration by presenting tables that other designers have made. ....	111
Figure 5.11 The Furniture Design Critic points out that her chair has only one armrest. It makes graphical annotation on the positions of armrests.....	113
Figure 5.12 The Furniture Design Critic suggests more design exploration. Claire decides to keep this chair design. ....	114
Figure 5.13 The Furniture Design Critic interprets Claire's chair design, introduces a concept of leg room, and points out chair's braces cause discomfort. It also presents an image to explain why braces are problematic. ....	115
Figure 5.14 The Critic suggests more design exploration.....	116
Figure 6.1 The 'dining set' task that the program has assigned in the third critiquing session.....	120
Figure 6.2 The Critic's reasoning process for selecting critiquing methods (Vignette 1) .....	123
Figure 6.3 The reasoning process of the Critic in Vignette 1 .....	125
Figure 6.4 The Critic's reasoning process for the selection of critiquing methods (Vignette 2) .....	127
Figure 6.5 The Critic's reasoning process in critiquing Vignette 2, scene 1 .....	128

Figure 6.6 The reasoning process in critiquing scene 2 (Vignette 2) .....	130
Figure 6.7 The reasoning process for the selection of critiquing methods (Vignette 3).....	132
Figure 6.8 The reasoning process in critiquing scene 1 (Vignette 3) .....	136
Figure 6.9 The reasoning process in critiquing scene 2 (Vignette 3) .....	139
Figure 7.1 Furniture Design Critic system components in the iterative construction-critiquing-repair cycle.	142
Figure 7.2 User Interface of Furniture Design Critic Program: Sketch window (left); Model window (right). .....	143
Figure 7.3 Sketch of a table with five legs.....	144
Figure 7.4 A five-legged table: this diagram shows part names (in black) and geometric elements: vertices (in red) and edges (in blue) .....	145
Figure 7.5 A book case design: this design violates a constraint that checks whether the back is large enough to resist lateral loads. The system offers feedback using the demonstration delivery type. ....	149
Figure 7.6 Pseudo-code and diagram showing a violated constraint. ....	150
Figure 7.7 Pedagogical Module and the data involved for selecting critiquing methods.....	153
Figure 7.8 A User Model Window displays the inferred data about the current user.....	164
Figure 7.9 Selected delivery types and communication modalities. ....	164
Figure 7.10 A state represents the delivery types and communication modalities used at a particular juncture in design critiquing.....	166
Figure 7.11 Path of Critiques: what critiquing methods were used and whether they were effective .....	167

## List of Tables

Table 2.1 Summary of Delivery Types in Critiquing Systems (•: Used, —: Not Used).....	49
Table 2.2 Summary of Communication Modalities in Critiquing Systems (•: Used, —: Not Used) .....	51
Table 2.3 Reviewed Critiquing Systems.....	52
Table 3.1 Factors for context-sensitive critiquing: Methods and Conditions .....	58
Table 3.2 Learning Styles and Disciplinary Difference .....	72
Table 3.3 Key Factors of Design Critiquing .....	76
Table 5.1 Space of Delivery Types and Communication Modalities .....	94
Table 5.2 Feedback Instances of Five Delivery Types.....	95
Table 6.1 Summary of the first critiquing session, “Ann” .....	122
Table 6.2 Summary of the second critiquing session, “Ben” .....	126
Table 6.3 Summary of the third critiquing session, “Claire” .....	131

## Abstract

Critiquing is a fundamental part of design education, yet we lack a clear and systematic understanding of how effective teachers make decisions about how to critique students. Although there is a considerable literature on design education, little has been written about design critiquing, specifically about critiquing strategies.

The dissertation outlines a theoretical framework of design critiquing practice developed through a literature survey. It then describes a computational model based on this framework, implemented in the Furniture Design Critic program, a kind of constraint-based tutor. The Furniture Design Critic provides a basis for describing and articulating critiquing strategies. The program first assesses the conditions of critiquing: how much a designer knows, his or her weaknesses and strengths, what critiquing methods have been effective for the designer, and the history of interaction between critic and designer. Based on this the Furniture Design Critic then selects a set of critiquing methods.

This program offers a computational model to describe design critiquing and to model inference about critiquing, and an environment for exploring and investigating alternative critiquing strategies. The dissertation contributes to an ongoing discussion of critiquing in design, design education, and intelligent tutoring systems.

# 1. Introduction

Consider a typical scene in a design studio in an architecture school. A studio teacher and a student are sitting around the student's work desk. Drawings, rough sketches, and diagrams are scattered around the desk along with several physical models. In a small notebook, the student has written a list of design concepts and ideas, illustrated with simple diagrams. The conversation begins with Quist (the teacher) looking at Petra's (the student) drawings and models<sup>1</sup>. Quist asks Petra what she is doing and inquires about the design choices she has made.

Petra has been designing a small residence. She describes her main design ideas, referring to her notebook. One of the main ideas is to draw sunlight into a living room by making slits and holes in the roof and walls. To explain how she has applied this idea in her residence design, the student

---

<sup>1</sup> I have taken the liberty of adopting the names of the student and teacher used in Donald Schön's well-known account of an architectural desk crit.

presents her drawings — a rough perspective of the living room with plans and sections—and physical models.

Quist rephrases Petra's words to confirm his understanding of her ideas. Quist also asks a series of clarifying questions, and Petra articulates her ideas, responding to her teacher's comments. Looking at Petra's drawings and models, Quist thinks aloud: *"Your main idea is to draw sunlight into this living space through these windows. You've placed the living room in the back part of the site, so the room might be a bit dark. But you want these windows to provide sunlight."* Then Quist asks a question to introduce a new idea: *"Have you thought about the path of the sun over a day and over a year?"* Petra (who is a bit naïve) says she has not, so she starts to think about how to improve her design in consideration of the sun's path.

But she looks puzzled, and Quist thinks the student is having trouble incorporating the idea into her design. He then refers to some famous architects' solutions to a similar problem: *"Do you know (architect) Steven Holl's chapel at Seattle University? How the building is placed in the site? Holl controls light with various shaped windows and the irregular shapes of the roof. Le Corbusier used a similar solution at Ronchamps, and Holl adopted Le Corbusier's design. They both designed the windows to control the quality of light; color, direction, and shape. So look at how Corbu and Holl used the shapes and sections of the windows to accomplish their design goals."* Quist then demonstrates how Petra can design her windows differently by dividing the roof into several different masses and by inclining a window to allow sunlight to enter the living room from a particular direction. As Quist explains, he draws perspective diagrams and section drawings. He gestures with his pencil, pointing to specific parts of the design and to draw Petra's attention to the inclined section of the window. At times, Quist evaluates Petra's work directly,

pointing out problematic parts or errors and praises parts of the design that seem especially promising. At the end of the critiquing session with her teacher, Petra is left with many ideas to improve her design and specific historical references to study.

Here I have described how a studio teacher (a human critic) might offer what is known in architectural education as a “desk crit” on a student’s work. The teacher uses various critiquing strategies during the session. He asks questions to lead the student to think about considerations she may not have thought of before, or he sometimes offers his opinion directly. While reviewing her drawings and models, the teacher might interpret the student’s design or may introduce new ideas. He might suggest a new approach, show examples, or describe how others have solved similar problems. He might demonstrate potential design solutions by altering the student’s model or directly evaluate the student’s work, either negatively or positively. In short, in everyday architectural design education design teachers use many ways to formulate and deliver critiques.

Design students in a studio setting are subjected to a series of critiquing sessions, especially one-on-one critiquing sessions, in which the studio teacher educates students by commenting on the work and helping them improve their designs. This critiquing process is an essential component in design teaching and learning (Schön 1985; Boyer and Mitgang 1996; Goldschmidt 2002), but no systematic study has set out to understand critiquing practice in design. With no accepted critiquing pedagogy, design studio teachers depend on personal experience from their own education or on intuition. Weaver, O’Reilly, et al. (2000) call this ‘hit-and-miss’ teaching.

I envision a computer-based critiquing system that could offer critiques to help individual students learn to design, much as human teachers do now in offering their desk crits. A computer program could lead a designer to develop better design solutions by offering timely and appropriate feedback to its users using diverse critiquing methods, just as a human critic does. However, at this time our understanding of design critiquing is too incomplete to implement this vision. We know little about the pedagogical strategies of expert design teachers—how they decide how to critique a student’s design. Today’s computer based critiquing systems also are quite primitive compared to human critics. Specifically, most critiquing systems offer only negative feedback in text, whereas—as we have seen—human critics use a diverse array of critiquing methods to comment on their students’ work. Thus, these systems are not yet able to support design and design learning and they do not provide a framework in which to articulate design critiquing.

This dissertation describes the Furniture Design Critic, a system I built to investigate how we might construct more versatile computer-based design critics. Needless to say, the Furniture Design Critic program falls far short of the performance of a human studio teacher. However, the framework that underlies the program provides a way to explain and understand design critiquing. In the future, it could serve as a basis for a more robust and comprehensive computer-based critiquing system. Specifically, the framework accounts for how to select critiquing methods based on conditions such as a student’s knowledge level, weaknesses and history of interaction with the design critic.

This first chapter offers an overview of the work. Important terms used throughout this dissertation are introduced in Section 1.1. Section 1.2 outlines the motivations for this research.

Sections 1.3 and 1.4 describe the scope and approach. Section 1.5 provides a guide to the rest of this dissertation.

## **1.1. Definitions of Terms**

### **1.1.1. Critiquing Methods and Conditions**

This section introduces two key terms: critiquing methods and conditions. Although I will explain and analyze these critiquing methods and conditions in detail later (see Section 3.2), I introduce these terms briefly here, because they will be used throughout this work. I define ‘critiquing methods’ as the various ways that studio teachers use to convey their design knowledge to their students. Teachers select particular critiquing methods, for example, organizing a certain critiquing setting such as a desk crit or a group critiquing session, or selecting a certain delivery type, for example, introducing new ideas or approaches, or demonstrating how to solve design problems. Among these critiquing methods the Furniture Design Critic program focuses on two critiquing methods, ‘delivery types’ and ‘communication modalities’. Delivery types are defined as ways that studio instructors deliver design knowledge: interpretation, introduction/ reminder, example, demonstration and evaluation. Communication modalities are defined as main communication channels: written comments, graphic annotations, and images.

I also define ‘critiquing conditions’ as the contexts in which critiquing occurs. Studio teachers consider these conditions when selecting a particular critiquing method. For example, the studio teacher considers the student’s knowledge, the design artifacts that the student presents for review, and the history of prior conversations between the teacher and the student.

## 1.2. Motivations

### 1.2.1. Lack of Understanding of Design Critiquing

Schön (1985) discusses design critiquing in his widely cited book, *The Design Studio*. To describe what design knowledge is conveyed and what a studio teacher does in a desk-crit to help a student improve a design, Schön uses the concept ‘*repertoire*’— a collection of images, ideas, examples, and actions. As professionals, designers build up repertoires from their experience. When a studio teacher reviews a student’s design, the teacher scans his repertoire for similar situations, for example, buildings he knows, or previously encountered problems. The teacher then shares knowledge drawn from his repertoire. The teacher seldom merely points out errors; he also describes examples, references similar situations from personal design experiences and demonstrates how to solve the design problem. Feedback presented using a variety of critiquing methods helps students understand their problems, eliminate errors from their proposed solutions, and eventually construct their own repertoire (Schön 1983; Uluoglu 2000).

Although Schön’s description of design critiquing is a helpful account for those unfamiliar with design education, in this account the details remain hidden and, for the most part, poorly articulated. Schön does not explain how studio teachers select methods for delivering a critique: what conditions they consider and how they decide which critiquing methods to use under what conditions. My literature review of design education (presented in Chapter 3) failed to uncover any studies in which studio teachers explain, in any detail, how they go about critiquing student work. In short, although the importance of the critique in design education is widely acknowledged, few have

studied the specific methods that teachers use to deliver and communicate critiques. This, then, is the focus of the research.

### **1.2.2. Limitations of Computer-based Design Critiquing Systems**

The work is motivated in part by the limitations of existing computer-based design critiquing systems. Although computer-based critiquing systems have been developed in a number of domains including design, current systems are still quite primitive. Most critiquing systems have focused on detecting errors, or opportunities for offering corrective feedback; few have investigated the methods with which critiques may be given. Specifically, no computer-based critiquing system for design supports various combinations of multiple critiquing methods. The Furniture Design Critic program presented in this dissertation addresses this gap in the existing literature. It proposes a framework for critiquing and an associated system architecture as a step toward building more sophisticated automated design critics.

A computer-based design critiquing system analyzes a proposed solution and offers critiques to help a user improve a design solution (Silverman 1992; Robbins 1998). The critiques help a designer identify problems and shortcomings in the design that must be resolved. Design critiquing systems have been built to support design in such various domains as architecture (e.g., ICADS and Design Evaluator) (Chun and Ming-Kit Lai 1997; Oh, Do et al. 2004), software engineering (e.g., Argo and DAISY) (Robbins and Redmiles 1998; Souza, Oliveira et al. 2003), civil engineering (e.g., SEDAR) (Fu, Hayes et al. 1997), programming (e.g., Lisp Critic and Java Critiquer) (Fischer 1987; Qiu and Riesbeck 2004), and medical treatment planning (e.g., TraumaTIQ) (Gertner

and Webber 1998). A review of these and other computer-based critiquing systems appears in Chapter 2.

Compared to a human critic, these existing computer-based critiquing systems deliver feedback in a quite restricted manner. Most systems provide only negative evaluations that point out problems; and most use only text to communicate these evaluations. These negative critiques only help designers identify design issues that they must resolve. In contrast, as we have seen, studio teachers employ a richer and more diverse set of critiquing methods. They interpret the student's design, introduce new ideas, demonstrate and give examples, and offer evaluations (Uluoglu 2000; Bailey 2004). They communicate using spoken and written comments, graphic annotations, and images (Schön 1983; Anthony 1991).

A few computer systems have supported different critiquing methods such as providing argumentation (Fischer, McCall et al. 1989) or offering design precedents (Nakakoji, Yamamoto et al. 1998). However, no prior research has attempted to support multiple methods of critiquing. This dissertation advances the field by demonstrating a system that selects among delivery types and modalities to offer critiques by considering the specific conditions at hand—the designer's knowledge, specific strengths and weaknesses, and the methods of critiquing that have previously proven effective.

### **1.3. Research Scope**

One might suppose that because it is based on the literature of design education and intelligent tutoring systems, the primary purpose of the Furniture Design Critic described here is to support

design learning. In a broad sense of course it is. However, I am not—in the first place—concerned with answering the question: “What is the best method of critiquing?”, nor in constructing a model to predict which critiquing methods will be most effective for which students under which conditions. Before that project can be undertaken, it is necessary to establish an explicit framework that can represent such a model. Establishing that framework is the first goal of this research.

The second goal is to demonstrate that, and how, a computer can be programmed to use the explicit framework to select methods for delivering critiques in a computational design environment. Making the framework operational in a computer program serves two purposes: first, to demonstrate that the framework can be used to build computer-based critiquing systems; and second, to provide a test-bed for experimenting with alternative critiquing strategies. I intend the Furniture Design Critic and the framework it entails to provide a foundation for investigating the pedagogical effectiveness of computational critiquing.

The focus and scope of this research is on the *strategic*, not the *domain* aspects of critiquing. Much of the past work on computer-based critiquing has focused on how to represent domain knowledge in a form that a computer can use to detect problems, or to reason about problems in the domain space. Instead, for the project described here, the specific problem space is incidental; it could be almost any physical design domain. Therefore I have chosen to illustrate the ideas in a fairly simple domain: The Furniture Design Critic program supports tasks in flat-pack furniture design, a problem domain that is commonly used to teach novice designers. Although the Furniture Design Critic is limited to this simple domain, the critiquing framework and the system for describing critiquing strategies that the program embodies can support much more complex domains.

In short, this program is a research tool to develop and test a framework for design critiquing, providing a means to represent critiquing conditions and to manipulate and select critiquing methods. The test of the Furniture Design Critic is not whether it helps students learn more effectively. The test of this tool is its ability to represent and implement alternative critiquing strategies. Thus it is not my intent to investigate which pedagogical strategies might be more effective than others. That could be the topic of future research using the framework and system presented here.

#### **1.4. Approach and Contributions**

The research described in this dissertation encompasses two main steps:

The first step is a comprehensive literature review. The review covers existing critiquing systems, intelligent tutoring systems, and conventional critiquing practice in design education. In order to develop a framework for critiquing I analyzed conventional critiquing practice as described in the literature in architectural design education. From this literature, I identified critiquing methods and critiquing conditions, and I propose a process model of *context-sensitive* critiquing. From the literature of intelligent tutoring systems I adopt the constraint-based architecture of Mitrovic (Ohlsson 1996; Mitrovic, Martin et al. 2007). And from the literature on computer-based critiquing I identified gaps in the performance of existing systems. This dissertation advances the field by developing a framework for multimodal context-sensitive critiquing.

The second step is the implementation of the Furniture Design Critic system. I implemented the Furniture Design Critic to demonstrate multimodal context-sensitive critiquing.

This program selects particular critiquing methods by considering the immediate context—the designer’s knowledge, strengths and weaknesses, and prior history. In building the Furniture Design Critic, I have investigated computational methods for selecting critiquing methods within this framework.

## **1.5. Guide to the Dissertation**

The dissertation is organized as follows: Chapter Two explains the context of this research by reviewing the literature of computer-based critiquing. In particular, it outlines the essential features of computer-based critics: (1) the critiquing process; (2) rules; and (3) intervention techniques. Chapter Three presents a review of architectural design critiquing practice as a descriptive framework of critiquing. First, I identify the fundamental factors for context-sensitive critique and based on these, I develop a process model of critiquing. Second, this chapter also discusses the advantages and disadvantages of individual factors. Chapter Four reviews the two main system architectures of intelligent tutoring systems: model tracing tutors and constraint-based tutors. This chapter compares these two architectures and explains why I chose to adopt the constraint-based architecture. Chapter Five introduces the Furniture Design Critic program using three examples. Chapter Six presents the computational mechanisms the program uses to select critiquing methods. It illustrates these mechanisms by showing how the Furniture Design Critic supports the example presented in Chapter Five. Chapter Seven describes the implementation details of the Furniture Design Critic. It outlines the system architecture, and describes what individual components do. Finally, Chapter Eight summarizes this research and outlines areas for future work.

## 2. Computer-based Design Critiquing Systems

Critiquing is the act of offering feedback or comments on proposed design solutions in progress. Teachers critique students' work to help them improve their solutions. Similarly, computer-based systems generate critiques to help users improve their designs and learn to solve design problems. Researchers have built systems to support designing in diverse domains such as medical treatment planning, architecture, software engineering, and programming. Most critiquing systems focus on finding errors in the user's proposed design solution. Silverman (1992) offers the following definition: *"Expert critiquing systems are a class of programs that receive as input the statement of the problem and the user-proposed solution. They produce as output a critique of the user's judgment and knowledge in terms of what the program thinks is wrong with the user-proposed solution."* Silverman is mainly concerned with user's errors—in his view, the goal of a critiquing system is to correct a user's mistakes.

Others see critiquing systems a little differently. Fischer (1991) states: *“Critics operationalize Schön’s concept of a situation that talks back. They use knowledge of design principles to detect and critique suboptimal solutions constructed by the designer.”* Robbins (1998) explains: *“A design critic is an intelligent user interface mechanism embedded in a design tool that analyzes a design in the context of decision-making and provides feedback to help the designer improve the design.”* From this point of view, critiquing systems need not only offer negative critiques, but can also help designers improve solutions by offering “constructive” feedback. A design critiquing system is a computer-based tool that analyzes a work-in-progress and provides feedback to help a designer improve their design. It may point out errors or mistakes, offer argumentation, or praise promising aspects of the design.

This chapter reviews computer-based critiquing systems in terms of three main components: the process of critiquing, the rules the systems use to trigger critiques and techniques the systems use to decide when and how to intervene. Table 2.3 summarizes the twelve critiquing systems reviewed here.

I found that while critiquing systems have potential to support designers, existing critiquing systems have considerable limitations. The review reveals limitations on how systems present critiques, specifically what I call ‘delivery types’ and ‘communication modalities’. Each system is committed to specific critiquing methods, most often negative evaluation in text. In contrast, human critics use various methods, not restricting themselves to a single critiquing method. Also, existing systems cannot support design and design learning and thus do not provide a framework in which to describe and articulate design critiquing.

## 2.1. Critiquing Process

Every critiquing system I surveyed uses the same basic process: it detects critiquing opportunities and offers feedback. A critiquing process model comprises a series of steps that a system follows in order to generate feedback. Several process models have been proposed. For example, in a survey of critiquing systems, Robbins (1998) identifies five phases of a critiquing process: Activate – Detect – Advise – Improve – Record (Figure 2.1).

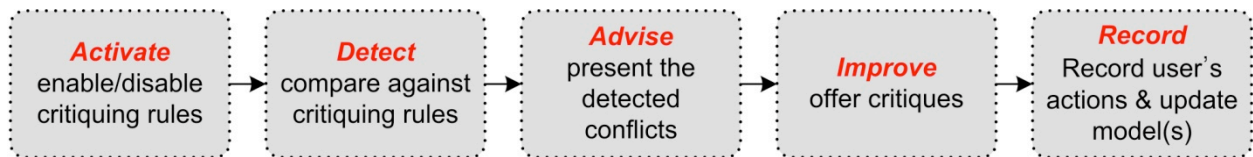


Figure 2.1 Robbins' Critiquing Process (1998): Activate – Detect – Advise – Improve – Record

In the **Activate** phase, a system enables and disables critiquing rules to support the user's current tasks and preferences. In the **Detect** phase, a system identifies opportunities by comparing the user's work with the system's critiquing rules. In the **Advise** phase, the system informs the user of detected conflicts. In the **Improve** phase, the system provides suggestions or advice about how to fix the problems. Finally, in the **Record** phase, the system records the designer's reaction to the critique the system offered. For example, if a user rejects a particular critique, the system may update its model of the user to avoid presenting the same feedback in the future.

Fischer and his colleagues (1998) suggest a slightly different process model specifically intended for design domains. This model includes the designer's action, referred to as the

**Create/Modify** phase, and it refactors the phases of the critiquing system. This model also emphasizes a design cycle (See Figure 2.2). A designer **creates** a design solution. The critiquing system **analyzes** the design and produces a **critique** for the designer to consider in the next iteration. Based on the offered critiques, the designer **modifies** his/her design solutions.

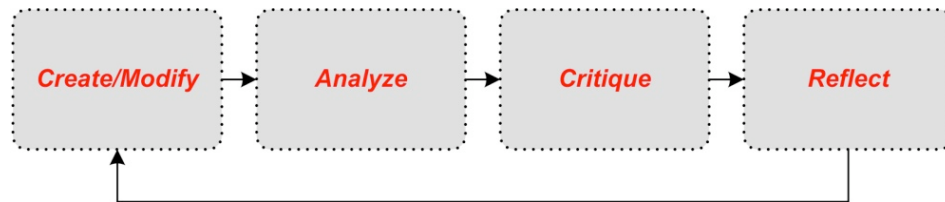


Figure 2.2 Fischer's Critiquing Process (1998)

## 2.2. Rules

Critiquing systems store design knowledge and use it to evaluate solutions. This section discusses four topics related to the storage of design knowledge that should be considered in the design and development of computer-based critiquing systems. These are (1) the form of design rules; (2) the completeness of the design knowledge; (3) the management of design rules; and (4) end-user rule authoring. The following sections elaborate on each topic.

### 2.2.1. Forms of Rules

All the critiquing systems I reviewed are rule-based; rules are represented in a classical expert system format (Fischer 1989). Each rule has a *Predicate* and an *Action*. *Predicates* represent particular conditions that may occur in design solutions. When the system detects a particular condition, that

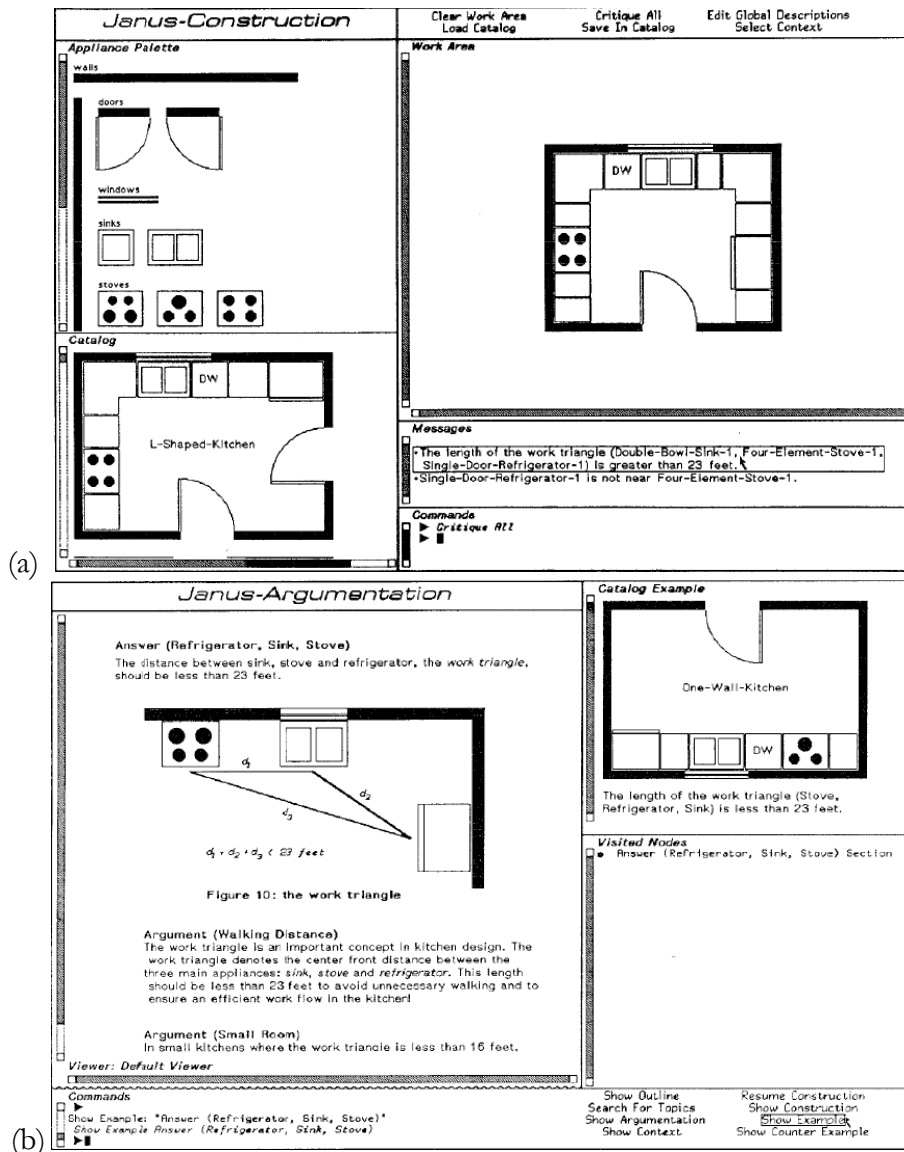
is, when a predicate matches a user's design, the critiquing system invokes the action. An action is the feedback the system gives in the form of argumentation, precedents, or negative/positive evaluation.

Gerhard Fischer and his colleagues at the University of Colorado developed a series of design critiquing systems in the late 1980s and the early 1990s. These systems, for kitchen layout design (Janus, Hydra, Crack, and KID; the Janus family for short) explored how to embed critiquing components in computational working environments. A rule in the Janus system (Fischer and Morsch 1988; Fischer 1989; Fischer, McCall et al. 1989; Fischer, Lemke et al. 1991) for kitchen layout design looks like this:

**Rule # : (left-of dishwasher sink)**

**Predicate:** If dishwasher is placed in the left side of sink

**Action:** then notify a conflict has been detected



**Figure 2.3 Janus Family:** (a) a user places kitchen appliances in the 2D working window and the system presents written feedback; (b) the system presents argumentation and positive or counter example kitchen layouts.

Figure 2.3-(a) shows the 2D working window where a designer places kitchen elements. Janus checks the layout against its rules and displays critiques as text in the message window. When

the user selects a critique, the Janus system reveals the underlying argument: It explains why the critique is important and what design issue, answer, and arguments it is associated with<sup>2</sup> (Fischer, McCall et al. 1989). Figure 2.3-(b) shows positive or counter-example kitchen layouts that the system presents on request. Janus depends on the user to identify the design goal (such as designing a family kitchen or a kitchen for one person), by filling out a form. Janus activates only rules relevant to the specified goals.

### **2.2.2. Completeness of Knowledge (Comparative and Analytic Critiquing)**

Existing systems employ what Robbins calls ‘comparative critiquing’, ‘analytic critiquing’ or both (Robbins 1998). In comparative critiquing, the user specifies a problem and then the system develops solutions by applying domain-specific rules. Then it compares the solutions it generated against the user’s work and reports the differences between them.

TraumaTIQ (Gertner and Webber 1998), a system that supports physician’s treatment planning for trauma, uses comparative critiquing. The physician first proposes a treatment plan. Next TraumaTIQ infers the physician's goals by reviewing the plan. The system then generates its own treatment plan, which it compares with the physician's. The resulting evaluation identifies three

---

<sup>2</sup> Raymond McCall integrates the ‘argumentative approach’ with a graphic design environment in the Janus system (Fischer, McCall et al. 1989). The idea is to augment the designer’s ability to perceive breakdowns from the proposed solutions and to present argumentation to support reflection about the breakdowns. Horst Rittel’s argumentative approach to design intends to improve design by supporting the designer’s reasoning rather than automating it. To implement this approach Rittel proposed IBIS (Issue-based Information System) method for documenting design discussion. IBIS organizes design discussion around the deliberation of issues that are design questions. Deliberation is the process of arguing the pros and cons of proposed answers to the issues. The components of IBIS are issues, answers, arguments, and resolutions. A resolution is a set of decisions to accept or reject answers to an issue. During deliberation, issues are raised, answers proposed, and arguments are given for or against the various answers. An issue is resolved by selecting answers based on the argumentation. In the Janus system, each rule has these components (e.g., issues, answers and arguments) to provide designers with argumentation.

types of errors: omission (a goal is not achieved), unexpected action (achieving the goals had a bad side-effect), and scheduling (steps are not performed in the recommended order). TraumaTIQ classifies error impacts at three levels: tolerable, probably harmless; non-critical, but potentially harmful; and critical, potentially fatal. The system generates critiques by filling templates with appropriate words to form sentences and presents critiques either as information or as a warning, depending on the level of error impact.

Comparative critiquing requires complete and extensive domain knowledge in order to generate good solutions; the knowledge-base must be able to represent all the designer's possible actions and solutions. This approach is reasonable for critiquing in well-structured domains such as high-school algebra. However, in many other interesting domains, in particular design, it is impossible to pre-formulate all possible actions and solutions. For these domains comparative critiquing is not an appropriate approach. It limits exploratory problem solving, because computer-generated solutions suggest what the user is supposed to do. Design problems seldom have one definite "right" answer; instead, a design problem may have several or even many "right" answers. A system that reports the differences between the solution it has generated and a user's proposed design would discourage the user's exploration of potentially valid other alternatives. The differences reported by the comparative critiquing system would lead a designer to examine why her solution differs from the system's solution, whereas her solution may be just as good, or even better.

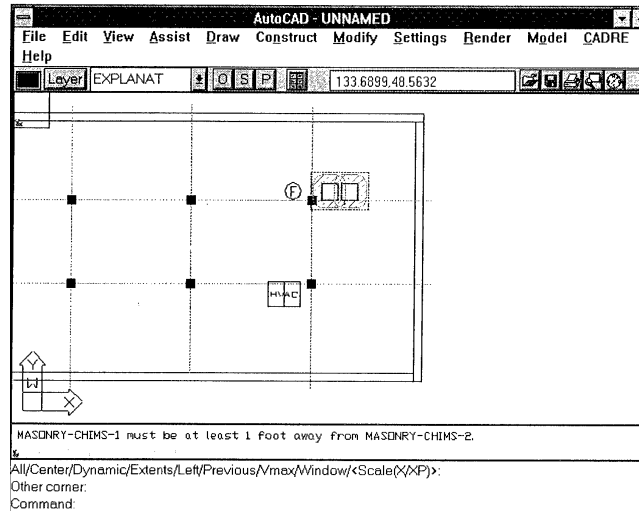
More appropriate to design critiquing is the analytic approach. In contrast with comparative critiquing, the analytic approach can be applied to a broad range of domains in which domain knowledge is incomplete (Robbins 1998). Analytic critiquing requires only that the system has

sufficient knowledge to detect possible critiquing opportunities; it does not require enough knowledge to generate a solution. Instead of comparing the user's solution against a system-generated right answer, an analytic critic analyzes the user's solution using stored domain rules. The Janus family of design critiquing systems uses the analytic approach. Janus offers feedback by checking a user's layout against design rules, although it cannot develop valid kitchen layouts. The Furniture Design Critic system developed here adopts the analytic critiquing approach.

### **2.2.3. Rule Management**

I turn now to how systems manage rules to offer relevant and timely feedback according to users, tasks, and goals. Most critiquing systems use specific representations to control the activation of rules (critiques). They use (1) a model of the task that the user is engaged in; (2) a model of the particular user who is doing the task; and (3) a model of the goals the user must achieve.

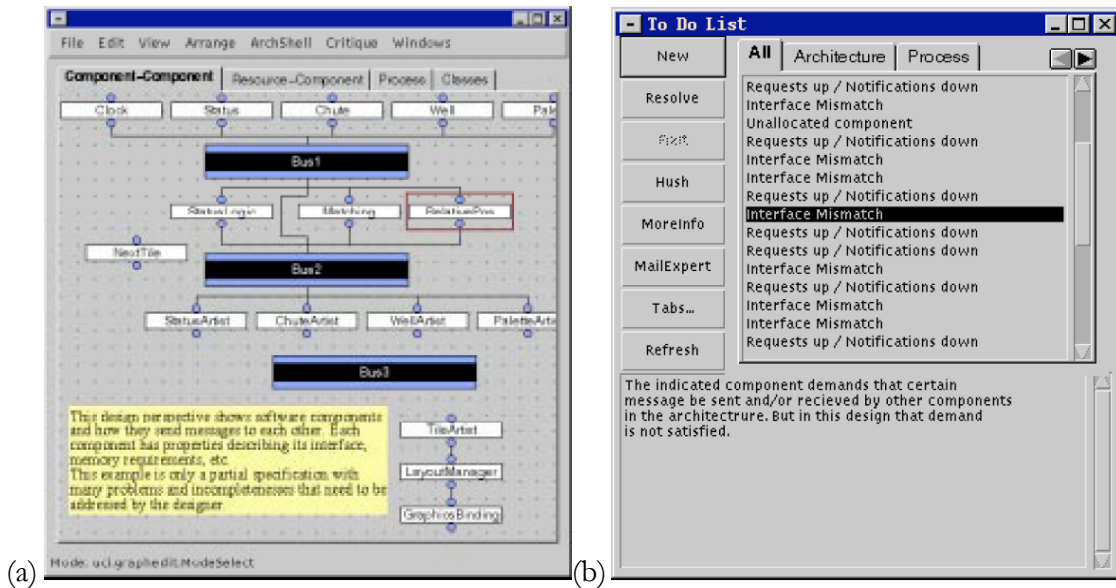
It is helpful if a critiquing system knows what the user is trying to accomplish. The purpose of a *task model* is to provide critiques that are relevant and timely to the task at hand. The SEDAR (Support Environment for Design and Review) system critiques roof designs based on constructability; for example, it checks to see whether a mechanical unit on a roof is placed in an illegal or unsafe position (Fu, Hayes et al. 1997). Using a built-in task model constructed by the system designer, SEDAR models tasks such as roof component layout and equipment layout in a hierarchy and it traces the designer through the task model based on the designer's actions. It infers which rules are relevant to the designer's current task and provides task-relevant feedback. It presents feedback in text, for example "*masonry chimney1 must be at least 1 foot away from masonry chimney2*" and circles relevant parts in red on the CAD drawings (see Figure 2.4).



**Figure 2.4 SEDAR:** the system offers textual critiques in the bottom window, “masonry chimney1 must be at least 1 foot away from masonry chimney2” and highlights relevant design objects in color

Another design critiquing program that uses task models is the Argo system for software engineering (Robbins and Redmiles 1998). The designer must explicitly identify the current task (e.g. system typology, component selection, etc.) from a list. The system activates only rules that match the selected task. This system then checks the designer’s software design using the selected rules. If the designer indicates that she is working on “rough organization”, then Argo would not activate a critique related to details. Of course, if the designer indicates the current task incorrectly, the system cannot provide a relevant critique.

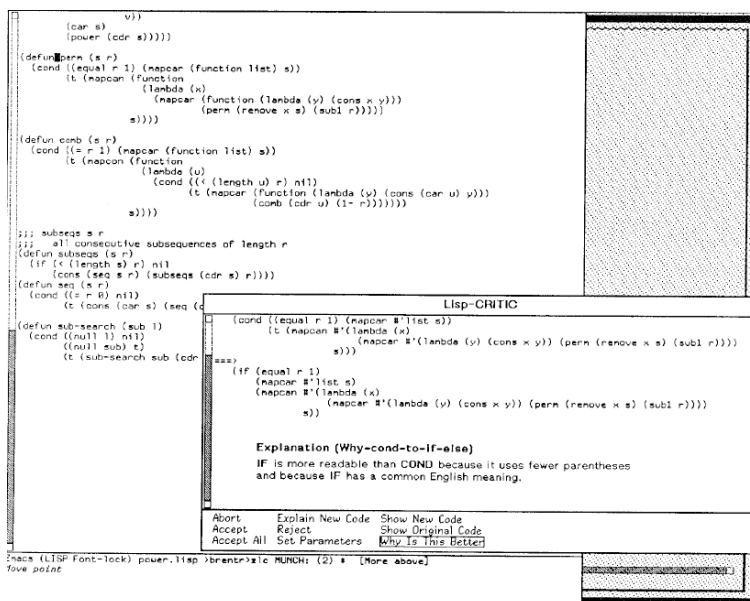
Figure 2.5-(a) shows the Argo window in which the user constructs UML (Unified Modeling Language) diagrams that represent software components and their interactions. Figure 2.5-(b) shows the system’s feedback in the To Do List window at the right.



**Figure 2.5 Argo:** (a) a user draws UML diagrams that represent software components and the interactions between them; (b) Argo presents feedback in the To Do List window. When a user clicks the name of the critique, the system offers a detailed explanation of the error.

A *user model* enables the system to adapt to a particular designer's preferences, knowledge, and past actions. Feedback offered by a system that has a user model could differ for a novice designer or an expert. Lisp-Critic (Fischer 1987; Mastaglio 1990) supports a programmer by providing feedback and on-demand explanation of specific Lisp functions (Figure 2.6). Its user model represents a programmer's understanding of the Lisp programming language. When the programmer invokes the critic, the system analyzes the function definition that s/he is currently working on (i.e. at the cursor). The system suggests alternative ways to write the same function. The programmer can accept or reject the feedback and ask for an explanation of the system's recommendation. Lisp Critic's user model maintains information about the domain knowledge and the preferences of each individual programmer. This user model helps the system to customize

explanations and to determine which rules to activate for each user. For example, when Lisp Critic notices that a programmer prefers using one function to another equivalent one, it turns off the rule that recommends using the less-preferred function.



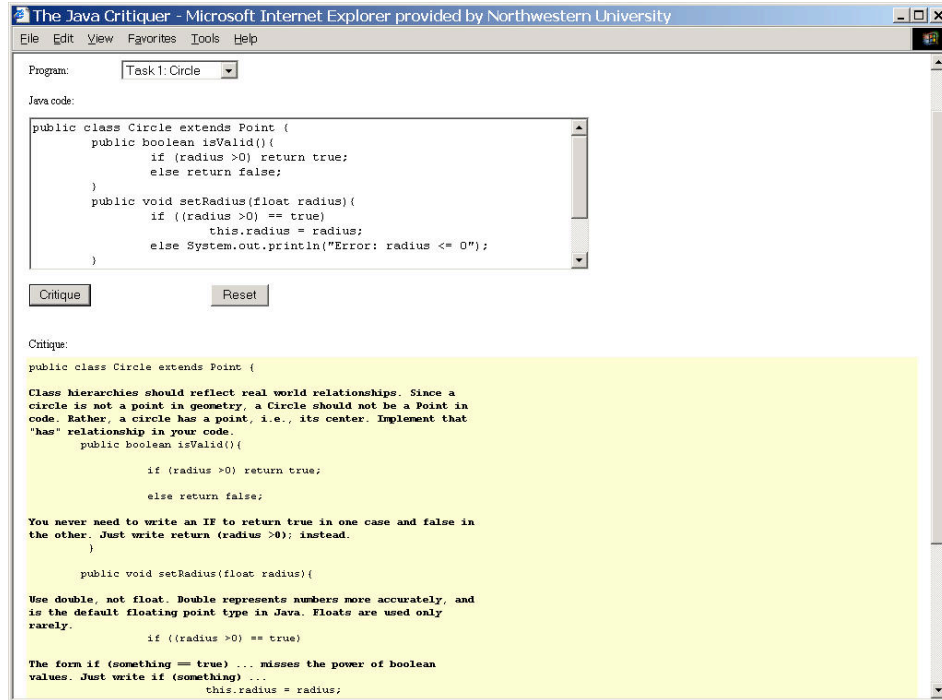
**Figure 2.6 Lisp Critic:** the large window is the user’s work environment. The Lisp-Critic window on top of the working window displays critiques. Here it presents a ‘cond-to-if’ transformation and explains why the system recommends changing the cond to the if expression.

Whereas a *task model* represents a set of subtasks and a structure of these subtasks, a *goal model* represents what a user is trying to accomplish. A user may create a goal model by specifying his/her task goals independent of developing design solutions. This tells the system what the user is trying to accomplish. For example, Janus activates only rules that are relevant to the goals specified by users. For example, if a user is designing a kitchen for one person, Janus deactivates rules for family kitchens.

#### **2.2.4. End User Rule Authoring**

In most critiquing systems, system designers write the rules in advance and end-users have no way to adjust the established rules or to incorporate new rules into the system. However, a designer may wish to change the scope and contents of rules. This has led several researchers to explore end-user rule authoring. Qiu and Riesbeck (2004) explored how users can author rules in Java Critiquer, web-based software that helps teachers critique students' Java code. A teacher loads a student's Java code and the system inserts critiques below any code that it finds problematic (Figure 2.7). The critique includes a short description, reasons for the critique, and suggestions, and the teacher can edit or delete the system's critiques. The teacher can also enter additional rules into Java Critiquer. This enables the teacher to gradually improve and extend the knowledge of the system.

Enabling end users to author rules can improve a critiquing system's accuracy, relevance, and scope. It enables users to store their own rules or principles as well as adapt to changing circumstances. Needless to say, users must take care when authoring rules. If a user adds a wrong rule the system will offer incorrect or irrelevant feedback.



**Figure 2.7 Java Critiquer:** this system supports programming teachers. A teacher can review, modify, add or remove system-generated critiques. The teacher can write new rules using regular expressions.

### 2.3. Intervention Techniques

Critiquing systems use various intervention techniques to provide users with feedback while they design. Robbins (1998) mentions a 'negative connotation' of critiquing systems: they can make designers uncomfortable by finding errors and faults in every design action. The timing, activation, modality of the critique and the types of feedback offered can play a role in mitigating the negative connotations of critique.

### 2.3.1. Timing

An important aspect of intervention is timing: *When* does the software offer critique? Fischer classifies timing as either **reactive** or **proactive** (Fischer 1989). Reactive critiquing offers feedback on work the designer has done. In contrast, proactive critiquing guides the designer by presenting guidelines before s/he begins to design. Silverman (1992) categorizes timing as **before**, **during** or **after** a task. Before-task critiquing corresponds to Fischer's proactive critiquing. During-task and after-task critiquing is reactive.

SEDAR uses all three timing strategies of Silverman's categorization: before (error prevention), during (design review critic, design suggestion) and after (error detection) (Fu, Hayes et al. 1997). Most building code checking systems (e.g., Solibri Model Checker and CORENET) in architecture provide *reactive* critiques *after* a designer has finished the work. The CORENET system (CORENET 2004) is used by Singapore government agencies to automate checking building plans. Architects and engineers submit CAD drawings via the Internet and CORENET checks the drawings against building codes and regulations. Based on the system's evaluation, revision requests are sent to the architects and engineers who submitted the plans. Another commercial system, the Solibri Model Checker (2007), imports 3D CAD drawings from major BIM (building information modeling) tools. Based on building codes, Solibri checks for potential design flaws and weaknesses. This checker also highlights any problems with the 3D model (Figure 2.8).



**Figure 2.8 Solibri Model Checker:** when a user selects an item (in this case, space ventilation) in the left window, the system presents its review, and then the system highlights relevant parts in the 3D model.

### 2.3.2. Activation

Fischer identifies two activation strategies: active and passive (Fischer 1989). Active critiquing continuously monitors the designer's moves and offers feedback when the system detects a critiquing opportunity. In contrast, passive activation provides feedback only when a designer requests it.

There is an interesting finding related to activation technique. Anderson et al. studied how users respond to active and passive activation critiquing (Anderson, Corbett et al. 1995) to learn which activation technique is more effective. The same critiquing system was provided with two different activation settings. In the passive setting, users did not ask for evaluation until they

completed a preliminary solution. In the active setting, users fixed errors immediately 80% of the time. From this we might conclude that active critiquing is better.

However, the question of when to activate critiquing is not so simple. A critic must consider various factors, such as how important is the error?, who is the designer?, what is the phase of design?, and so on. For example, if an experienced designer is just starting to explore design alternatives, a critic may choose to defer critiquing. Alternately, if a novice designer is in the early design phase, the critic might offer feedback immediately to protect the designer from serious errors that could influence the overall design.

### **2.3.3. Delivery Types**

Most critiquing systems offer negative evaluation that focuses on resolving mistakes. Some systems also offer praise, explanation, argumentation, examples (precedents), introduction of new information, and interpretation. Janus praises the good aspects of a kitchen layout, to encourage designers to retain the layout in further revisions (Fischer, Lemke et al. 1991). Some systems offer detailed explanations (Souza, Oliveira et al. 2003) and argumentation describing why particular parts of the design are desirable or problematic (Fischer 1989). For example, when a user's design violates the rule that represents the eating-cooking-cleaning "work triangle" concept, Janus explains the work triangle and why it is important in a kitchen layout. The KID (Knowing-in-Design) system provides kitchen layout examples as potential solutions that could cue further design moves (Nakakoji, Yamamoto et al. 1998).

To support software design CodeBroker (Ye 2003) retrieves Java methods and classes from an API library that users may not know and introduces them. The programmer can adopt these methods and classes without writing them from scratch. CodeBroker is embedded in the emacs editor that the programmer uses to write code. If a programmer plans to write a method to generate a random number between two integers, s/he describes this task in the documentation string of the Java method. Codebroker extracts the programmer's comment and uses it as a search query to retrieve components from the Java library, such as *getInt*, *getLong*, and *getFloat*. (The programmer also can refine the queries by modifying them.) The system retrieves library components such as classes and methods and presents them below the emacs buffer with that contains the code. In Figure 2.9 below, *getInt* is the best match for the user's task.

The screenshot shows the Emacs editor window titled 'emacs@buddy.cs.colorao.edu'. The menu bar includes Buffers, Files, Tools, Edit, Search, Mule, JDE, Java, and Help. The main buffer contains the following Java code:

```

/** This class simulates the process of card dealing. Each card is
    represented with a number from 0 to 51. And the program produces
    a list of 52 cards, as it is resulted from a human card dealer */
public class CardDealer1 {
    static int [] cards=new int[52];
    static {
        for (int i=0; i<52; i++) cards[i]=i;
    }
    /** Create a random number between two limits */
    public static int getRandomNumber (int from, int to) {

```

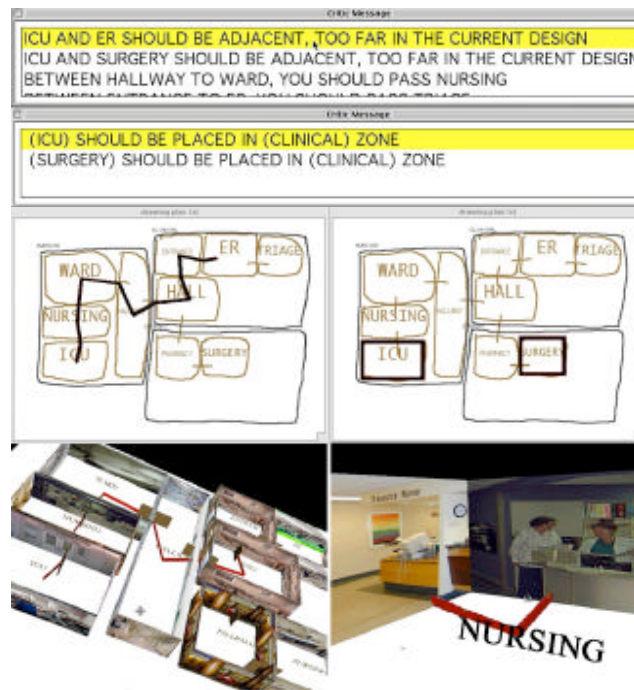
Below the code, a list of retrieved methods is displayed in a separate buffer titled 'CardDealer1.java (JDE)--L10--All--'. The list shows the following methods and their descriptions:

Rank	Score	Method Name	Description
1	0.69	getInt	Generate a random number using the default generat
2	0.64	getLong	Generate a random number using the default generat
3	0.59	getFloat	Generate a random number using the default generat
4	0.59	getDouble	Generate a random number using the default generat

At the bottom, the source of the top method is shown: 'com.objectspace.jgl.util.Randomizer::static int getInt(int lo, int hi)'.

**Figure 2.9 CodeBroker:** retrieves and offers library components such as classes and methods

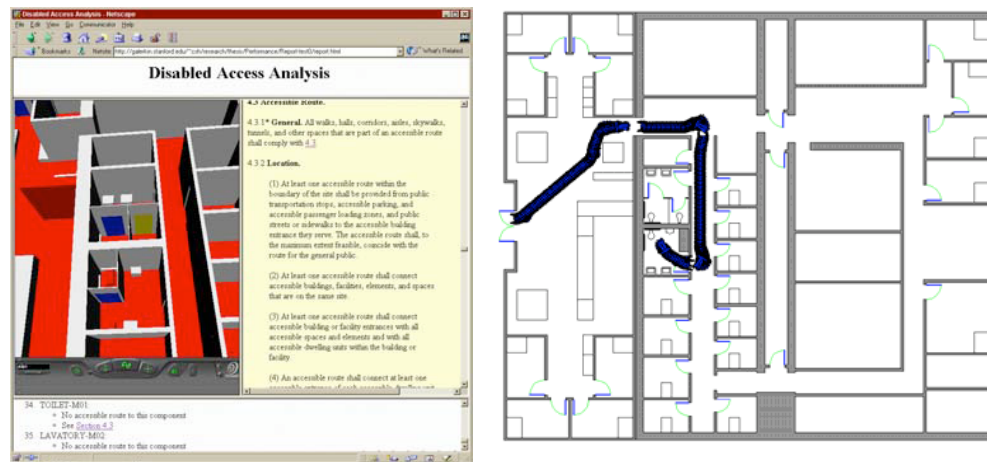
Some systems also interpret design solutions from a specific viewpoint; for example, Design Evaluator enables designers to sketch a floor plan diagram; then the system indicates problematic paths in a 3D VRML (Virtual Reality Modeling Language) model (Oh, Do et al. 2004). Figure 2.10 shows three ways that Design Evaluator offers critiques: textual critiques, graphical annotations, and 3D visualization.



**Figure 2.10 Design Evaluator:** the system offers feedback in several ways: text feedback, graphical annotations and 3D visualization (VRML model).

Similarly, Han's building code checker provides 3D VRML models of a floor plan, interpreting the building plan from the viewpoint of a disabled building user (Han, Law et al. 2002). The checker presents the evaluation results by showing simulations on VRML models (Figure 2.11).

To check the floor plan for disabled accessibility the system simulates a building user in a wheelchair. If the checker finds a certain room to be inaccessible, it displays this issue on the VRML models and CAD drawings.



**Figure 2.11 Han's Universal Code Checker:** the system checks the design against building codes and reports the evaluation results on the web pages by showing simulations on VRML models and graphic annotations on CAD drawings.

Table 2.1 summarizes the delivery types in the critiquing systems I reviewed. Most systems offer only evaluation (negative and/or positive) critiques. However, a few offer additional delivery types. Design Evaluator and Han's code checker interpret the proposed solution by presenting the building user's movements in the designed space. The Janus family of critiquing software offers example kitchen layouts when a user asks for them. CodeBroker introduces new ideas by providing usable components such as classes or methods.

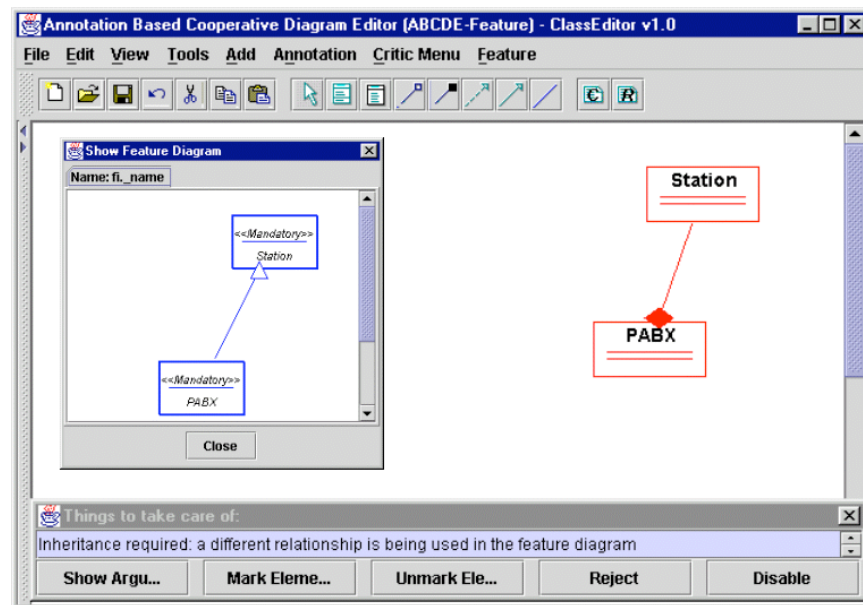
**Table 2.1 Summary of Delivery Types in Critiquing Systems** (•: Used, —: Not Used)

Systems	Delivery Types				
	Interpretation	Introduction/ Reminder	Example	Demonstration	Evaluation
Janus Family	—	—	•	—	•
Lisp Critic	—	—	—	—	•
CodeBroker	—	•	—	—	—
Java Critiquer	—	—	—	—	•
Argo	—	—	—	—	•
DAISY	—	—	—	—	•
SEDAR	—	—	—	—	•
TraumaTIQ	—	—	—	—	•
Design Evaluator	•	—	—	—	•
Solibri Checker	—	—	—	—	•
CORENET	—	—	—	—	•
Han's Universal Code Checker	•	—	—	—	•

#### 2.3.4. Communication Modalities

How should a system present a critique to the designer? Existing systems use one or more of three communication modalities: written messages (text), graphic annotations, and 3D visualizations. Most systems provide feedback in written messages. Several also augment text with visual critiques such as graphical annotations and images. For example, my own Design Evaluator system (Oh, Do et al. 2004) annotates a designer's floor plan diagram and generates a 3D view of the building with graphical annotations. Some building code-checking systems also illustrate results by circling

problematic parts on a 3D model (Han, Law et al. 2002). In the DAISY software design environment, the user draws UML (unified modeling language) feature and class diagrams which the system annotates (Souza, Oliveira et al. 2003). The system evaluates the user's diagrams and the consistency between feature and class diagrams. It offers critiques in text and shows the user the correct graphical notation (Figure 2.12).



**Figure 2.12 DAISY:** the user draws feature and class diagrams using UML. DAISY presents textual critiques in the “Things to take care of” window and displays a correct graphical notation beside the user’s diagram.

Several researchers emphasize the value of graphic annotations on drawings and 3D models. Based on usability tests, Fu (1997) argues that graphic annotations help the designer understand critiques better because designers find it difficult to relate text critiques shown in a

separate window to graphic elements in their drawings. Han et al. (2002) argues that 3D models can help designers predict the performance and behaviors of artifacts such as a wheelchair.

**Table 2.2** Summary of Communication Modalities in Critiquing Systems (•: Used, —: Not Used)

Systems	Communication Modalities		
	Text	Drawing (graphic annotations)	Images
<b>Janus Family</b> (Janus, Crack, Hydra, KID)	•	—	• (examples)
<b>Lisp Critic</b>	•	—	—
<b>CodeBroker</b>	•	—	—
<b>Java Critiquer</b>	•	—	—
<b>Argo</b>	•	—	—
<b>DAISY</b>	•	•	—
<b>SEDAR</b>	•	•	—
<b>TraumaTIQ</b>	•	—	—
<b>Design Evaluator</b>	•	•	•
<b>Solibri Checker</b>	•	•	—
<b>CORENET</b>	•	•	—
<b>Han's Universal Code Checker</b>	•	•	•

## 2.4. Summary

I have reviewed a number of existing critiquing systems according to three key features: (1) the process the system uses to produce a critique, (2) the rules that trigger and formulate critiques, and (3) the techniques that decide when and how to intervene. Table 2.3 summarizes the domains and some interesting aspects of the critiquing systems I reviewed.

**Table 2.3 Reviewed Critiquing Systems**

<b>Systems</b>	<b>Domains</b>	<b>Interesting Aspects</b>
Janus Family (Janus, Crack, Hydra, KID)	Kitchen design	Goal model (argumentation, perspectives), User model, Delivery types (praise, negative evaluations, case-based critiques), Kitchen design interface (templates with graphic objects of kitchen appliance)
Lisp Critic	Programming	User model
CodeBroker	Programming	Retrieve and deliver library components (Java API), User Model
Java Critiquer	Programming education	Rule authoring
Argo	Software engineering	Diagram editor, Task Model (Task selection)
DAISY	Software engineering	Diagram editor, Modalities (graphic annotation with written feedback)
SEDAR	Roof design	Plug-in Program in CAD, Task model, Timing strategies, Modalities (graphic annotation with written feedback)
TraumaTIQ	Medical treatment plan	Comparative critiquing, Written feedback using templates
Design Evaluator	Architectural floor plan + web page layout	Sketch interface, Modalities (graphic annotation with written feedback), 3D VRML model, General framework beyond domain-specific system, Conceptual design diagramming
Solibri Checker	Building codes	Check building codes on 3D CAD drawing, Commercial software
CORENET	Building codes	Singapore government's system, Checking building codes, Modalities (graphic annotation and a written report of errors)
Han's Universal Code Checker	Building codes	Checking building codes (universal codes), e.g. wheelchair access, Graphic annotation on 3D models

I also reviewed the delivery types and communication modalities that critiquing systems use to present feedback (See Table 2.1 and Table 2.2). These two tables show that computer based

systems offer critiques in limited ways compared to the rich array of critiquing methods that human critics employ. Specifically, most systems offer only negative feedback. Negative feedback may be effective, compared to other delivery types, to improve a design solution because it directly points out what is wrong and even tells the designer what to do next. However, if the systems keep offering designers negative feedback, designers may lose control over their process. For example, they may only attempt to fix the raised problems, instead of exploring other alternatives or thinking through other design approaches. Many designers also prefer to struggle with design problems by themselves rather than follow prescriptive instructions. Therefore, as Robbins (1998) points out, direct negative feedback may unproductively make designers feel uncomfortable.

In a study of human-robot interaction Cristen Torrey (2009) investigated how people respond differently to robots and how they react to different communication strategies. Her research shows that polite communication strategies have a strong positive impact. In contrast, direct messages have negative effects, because people perceive this as more controlling or insulting. In short, the particular methods that a system uses to present a critique are likely to influence designers' reactions. That is why I chose to focus specifically on critiquing methods, delivery types and communication modalities.

## **3.A Framework for Critiquing Practice in Design Studio**

How do design studio instructors critique student work? To investigate this question, I reviewed the literature of design education to better understand critiquing practice in education settings.

Based on a review of the literature in design education I construct a framework for design critiquing. This framework could inform teachers about the pedagogical choices they make while critiquing. It accounts for what teachers do to convey their design expertise to students, what critiquing methods they use, and the advantages and disadvantages of each method of critiquing. The framework is also the basis of the Furniture Design Critic software described in later chapters of this dissertation.

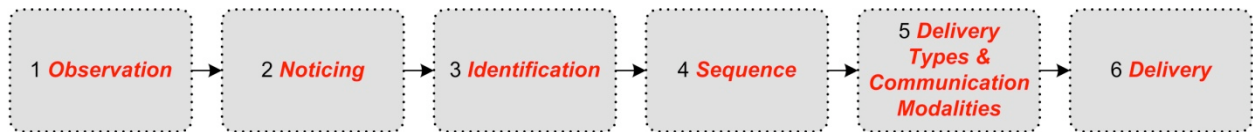
I focused on publications that pertain to architecture design studios. I examined major design journals (*Design Studies*, *Journal of Architectural Education* and *Journal of Architectural and Planning Research*), several key conferences (the Architectural Education Exchange, Design Thinking Research Symposium, and the Common Ground Design Research Society International Conference), a PhD dissertation, *The Digital Design Coach* (Bailey 2004), and several recently published books: Dana Cuff's book *Architecture: the Story of Practice* (Cuff 1991), Don Schön's *the Design Studio* (Schön 1985), and Kathryn Anthony's *Design Juries on Trial: the Renaissance of the Design Studio* (Anthony 1991). These published works variously present experimental data, protocol analyses, criticism and suggestions or proposals about design studio and critiquing.

This literature review offers a representative, though not necessarily complete, picture of critiquing in current architectural education. I analyzed and synthesized a range of observations and findings about critiquing. The next section presents a process model of critiquing for individual critiquing activity. I then discuss factors of design critiquing for context-sensitive critiquing.

### **3.1. A Process Model of Critiquing**

I can describe what happens during critiquing as a sequence of steps, or process model (Figure 3.1). When a student explains her design work by showing the studio instructor her drawings and physical models, the instructor listens and observes what the student has presented (*Observation*). Upon noticing problematic and promising aspects of the student's work (*Noticing*), the instructor must clearly identify the issues and why they are problematic or promising based on understanding the immediate learning goals (*Identification*). I separated the identification step from the noticing step, because identifying problematic or promising aspects of the student's work requires some

deliberation, whereas noticing may be done intuitively. The instructor then, considers the order in which to deliver feedback to the student (*Sequence*). For example, the instructor may decide to begin critiquing by pointing out positive aspects. Or, the instructor may address critical issues first and leave other less important issues for later.



**Figure 3.1 Critiquing Steps:** (1) Observation; (2) Noticing; (3) Identification; (4) Sequence; (5) Delivery Types and Communication Modalities; and (6) Delivery

After the sequence of delivery is determined, the instructor must decide on delivery types and communication modalities<sup>3</sup> (*Delivery Types and Communication Modalities*). There are many communication modalities the instructor can use to deliver comments to the student. The instructor may use hand gestures while talking, or may sketch over the student's drawing. The instructor must also formulate the critique in an appropriate manner, or what I shall call 'delivery type', (e.g., directive or facilitative). Finally, the instructor delivers the critique (*Delivery*). Here, linguistic choices and strategies are important (Torrey 2009). Subtle differences such as language choices, facial expressions, and voice qualities, influence the relationship between instructor and the student and hence the effectiveness of the critique.

---

<sup>3</sup> The Furniture Design Critic project described here focuses on this step in the process.

This analytical model describes a process for developing critiques and selecting delivery types and communication modalities through an analysis of the situation. It identifies the detailed steps of individual critiquing but it is not a cognitive model of critiquing, nor is a prescriptive model. I do not mean suggest that an instructor does, or must, follow these steps deliberately every time she critiques a student's design. However, the model seems to account in a general way for what human critics do, and it illustrates the foundation that underlies the Furniture Design Critic program presented in Chapter Five.

### **3.2. Fundamental Factors for Context-sensitive Critiquing**

How do studio teachers (critics) choose delivery types and communication modalities? What do they consider as they make critiquing decisions? To answer these questions, I have developed an overview of design critiquing practice as described in the literature of architectural education. This overview includes eleven factors that studio teachers may consider while formulating a critique. I divide these factors into two groups: methods and conditions (see Table 3.1). The five factors of *Critiquing methods* are critiquing settings, teacher-student relationship, communication modalities, delivery types, and delivery. The six factors of *Critiquing conditions* are design phase, individual differences, knowledge/experiences, students' response types, design artifacts, and learning goals. This section describes each of these factors and how they relate to one another.

**Table 3.1 Factors for context-sensitive critiquing:** Methods and Conditions

	<b>Factors</b>
<b>Critiquing Methods</b>	critiquing settings, teacher-student relationship, communication modalities, delivery types, delivery
<b>Critiquing Conditions</b>	design phases, individual differences, knowledge/experiences, students' response types, design artifacts, learning goals

### **3.2.1. Critiquing Settings**

Design education researchers refer to several types, or settings, of critiques that an instructor uses to interact with students. (I shall call them ‘settings’ to distinguish them from the term ‘delivery type’). Bailey (2004) provides the most comprehensive list: desk crit, group crit, interim review, final review, and informal interaction. These derive from his analysis of the history of architecture education as well as his interpretation of Schön’s observations (Schön 1985). The rest of this section explains these five settings of critiques and discusses three perspectives from which to examine them.

#### **3.2.1.1. Desk Crit(iques)**

A *desk crit* is an individual critiquing session involving an instructor and a single student often held at the student’s desk. Desk crits take place throughout the entire period (typically 12-16 weeks) of a studio course. Several researchers emphasize the value of desk crits in design studios. Schön (1983), Goldschmidt(2002), and Koch et al. (2002) all consider desk-crits to be an essential component of studio teaching. Koch et al. (2002) argue that desk crits are the most effective way for an instructor to monitor each student’s progress over time. Uluoglu (2000) notes that the desk crit enables an

instructor to lead individual students to see their design problems from the instructor's viewpoint. Goldschmidt (2002) found that desk crits transfer a wide range of critical design knowledge.

#### ***3.2.1.2. Group Crit***

Group crits engage a small group of four to six students. The instructor may schedule group crits frequently, as often as once a week. Students put work on the wall, or gather around each student's desk with the instructor and the instructor and students discuss each student's presented work. Group crits provide students the opportunity to see each student's approach to solve the same design problem.

Several studies point out the merits of group crits. For instance, Farivarsadri (2001 ) argues that group crits are especially appropriate for introductory design studios. They are valuable for students with little design experience because they expose students to multiple solutions to the same problem. Compared to larger reviews group crits tend to engage beginning students who may lack confidence to speak in a larger and more public session. Students can participate more actively in discussion because of the smaller group size and informal setting.

#### ***3.2.1.3. Interim Review***

Interim reviews involve the entire class at key milestones during a studio project. Instructors hold interim reviews when they think all students can benefit from sharing their progress and knowledge with others in the class or when the instructor sees that many students are encountering similar problems or opportunities in their designs. The first interim review often occurs after students have performed an analysis of the building site. Students share their analyses of site and data such as

historical background, urban conditions, neighborhood character, or environmental issues. Another common time for an interim review is as students prepare for their final review at the end of the studio course. Each student presents his solution to a small jury group composed of other studio instructors, professional architects, and sometimes even clients. This serves as a rehearsal for the final review. During interim reviews, while one student's work is critiqued, other students listen to the comments made by the instructor and external reviewers, and often provide their own comments.

#### ***3.2.1.4. Formal (Final) Review***

The final review has the character of ceremony or ritual in the design studio and some students dress up in formal or special attire. It is held at the end of the course and external critics are often invited. The "jury" of critics typically consists of three to five local architects, the instructors of other studios at the same school, other non-studio faculty members (i.e.: a structural engineer or an architecture historian), or representatives of the client, if there is one.

At the formal final review, students usually prepare a large panel where they arrange the key drawings that describe their designs. Students present their drawings and physical models as the jury moves from one student to the next, commenting on each work publicly. Jurors are sometimes asked to fill out an evaluation form for each student, which is later given to the student along with the studio instructor's assessment of his or her performance over the entire studio course.

Researchers note several goals for a final review. For example, Dinham (1986) suggests three purposes:

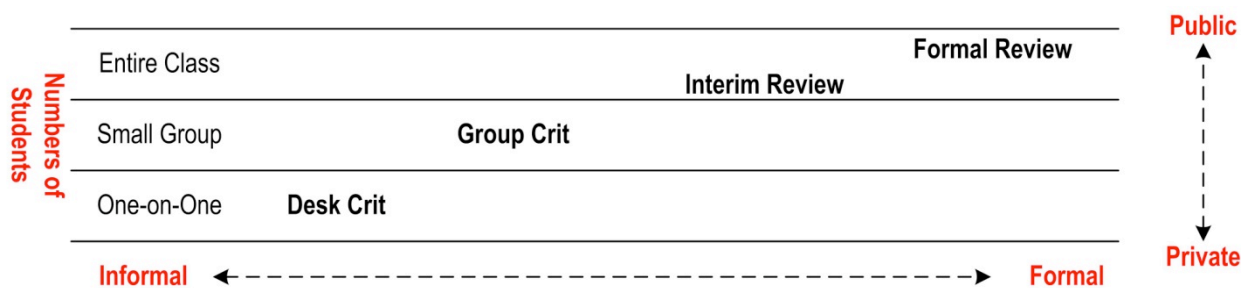
- a). The Jury can directly teach individual students by discussing and evaluating their designs.
- b). The final review is a tool for teaching all students in the studio together. As the jury comments on an individual student's work, they often broaden the scope of discussion from an issue found in one student's work to a common issue—leading other students to learn from the work of their classmates.
- c). A jury composed of professional architects who continually engage in professional dialogue, provides students the opportunity to hear challenging and inspiring conversation, observe the professionals' skills, and perhaps acquire some of their expertise. Students can learn the prevailing culture of architectural practice and professional experience. They themselves can practice analyzing and evaluating the presented projects while referencing their expertise or experiences, and observe how to conduct a professional presentation.

Despite its undoubted value, the formal review is often surrounded by what Argyris (1981) calls the “*mystery/mastery syndrome*”. The term “*mystery/mastery*” refers to the tendency of critics to use sophisticated words conveying an aura of mystery in order to display their mastery of architectural expertise. This tendency can confuse students and make it more difficult for them to understand the professionals' discussion and comments. Another criticism of formal reviews is that the conversation can intimidate students. Formal reviews with juries are characterized by their open and public nature (Anthony 1991). Juries may give serious and harsh criticism to students in front of the class. Interaction between jury and students may become hostile or adversarial (Groat and Ahrentzen 1996).

### 3.2.1.5. *Informal Interaction*

Design studios are organized to foster informal discussion among students. In an open studio space, students naturally monitor each other's progress, comment informally on one another's work, compare design approaches, or learn certain skills such as drawing and modeling (Cuff 1991). It is not unusual for students to benefit from overhearing other students' desk crits.

Cuff (1991) also reports that students are likely to form close learning groups after working together for long hours in a common studio space. Students benefit significantly from interacting with their peers as they share interests and problems (Schön 1985).



**Figure 3.2 Three Perspectives of Critiquing Settings:** (1) Numbers of Students; (2) Public – Private; and (3) Informal – Formal.

I have listed five settings of critique. As shown in Figure 3.2, we can discuss these settings from three perspectives: (1) Number of students, (2) Public / Private, and (3) Informal / Formal. First, we can characterize a critiquing setting by the number of participants. In general, the instructor can expect more active participation from students in smaller groups. Second, a critiquing setting can be identified on a continuum between private and public. Desk crits are highly private, although

in an open studio environment, students can easily hear another student's critique. The larger the group, the more public the critiquing session becomes, and external judges and reviewers make the critiquing session highly public. Finally, a critiquing setting can be characterized by its formality. Informal critiquing sessions tend to be more constructive and formal critiquing sessions more evaluative. Some students may be reluctant to voice their opinions in more public or formal sessions. Compared to group crits, interim reviews are deliberately more formal, and less peer participation is expected. Interim reviews are, in turn, less formal than a final review. The same jury members who attend an interim review may also attend the final review, but during an interim review their critiquing tends to be more constructive than evaluative (Anthony 1991; Bailey 2004). Highly regarded architects are often invited to the final review – adding to the formality. In addition, the final review often attracts a large audience of interested students and local architects who were not involved in the design project being presented.

### **3.2.2. Teacher-student Relationships**

In design studios students learn from comments from instructors and peers. Several researchers note that the relationship between the studio instructor and students may influence a student's learning. They identify three types of relationships and report on the merits and disadvantages of each. These relationships are: Master and Apprentice, User and Designer, and Peer Critiquing.

#### ***3.2.2.1. Master and Apprentice***

The most common model for the relationship between an instructor and a student in a design studio is the *master and apprentice*. The instructor plays the role of the *master* and the student plays the role of

*apprentice*.<sup>4</sup> The premise of this model is that the instructor (the master)—often a professional architect—holds the knowledge and experience to solve all the design problems the student will work on. Thus the instructor can provide helpful feedback as the student develops solutions for design problems.

Some researchers raise concerns about students feeling disoriented in their learning when their teachers act as a master. In the master model the instructor has power to control the student's work, so that it is easy for students to follow the instructor's direction without considering what they would like to develop or fully understanding what the instructor's feedback means. A student might follow the master's critiques blindly without integrating the master's feedback into his or her own thinking. For example, a student might just follow the approach demonstrated by the master without reflecting on or understanding the suggestions. Dutton (1991), Odgers (2001) and Koch, Schwennsen et al. (2002) all note that the overpowering master's feedback might impede the apprentice's learning and critical thinking. Grasha (1996), likewise, argues that the instructor who is frequently overbearing while displaying his expertise can intimidate students.

Several researchers propose alternative strategies to studio teaching. For example, Dutton (1991) recommends avoiding desk crits and conducting peer review sessions instead because the

---

<sup>4</sup> Instead of the term "master," Attoe and Mugerauer (Attoe and Mugerauer 1991) use a different term, "coach," but they describe a similar student-teacher relationship. They describe three roles for teachers: coach, counselor and parent. "Coach" is similar to the "master" model. A teacher as "counselor" plays a gentler role and aims to help students discover themselves. A counselor's goal is to lead students to think critically on their own. In the "parent" model, a teacher is more comprehensive: nurturing and open to the student's academic growth. Attoe and Mugerauer argue that coaching and counseling should be replaced by the parenting role. Compared to the first two roles, the main character of parenting is *caring*. They describe *caring* as showing an interest in the student's ' individual life as well as work.

overpowering authority of the instructor discourages students in a desk crit from participating freely in debate, asking questions, and reflecting on their own designs (Odgers 2001). To empower students, Odgers (2001) proposes replacing individual critiquing sessions with asynchronous communication between students and the instructor using a design journal. Students would be asked to record their design rationales, sketches and to describe their difficulties. Odgers argues that the journal would encourage students to participate in later public review sessions, such as interim and final reviews.

#### ***3.2.2.2. User and Designer***

Some instructors employ a *user–designer* relationship (Dutton 1991). In this relationship, the instructor refrains from judging the students’ work—offering neither positive nor negative evaluations. Instead, the instructor acts to represent a user or a group of users and comments on the design from the perspective of a user. Although it is impossible to eliminate the instructor’s role as an expert, the user-designer approach provides a less intimidating, more constructive learning environment.

#### ***3.2.2.3. Peer Critiquing***

Peer critiquing may occur either in informal discussions or in group crits. Students discuss personal experiences and viewpoints with their peers who are engaged in solving the same design problem. Although peer-critiquing sessions do not involve the instructor, he can play a role by providing students the opportunity to critique each other’s work, and by demonstrating how to critique appropriately. Several researchers detail the merits of the peer-to-peer critique (Parnell 2001;

Farivarsadri 2001 ; Bailey 2004). Looking at their peer's work exposes students to alternatives and approaches for the same design project. Peer critiquing also enables students to participate more actively in debates or discussions. Students learn to formulate a critique and to take responsibility for what they learn. And peer critiquing supports collaborative learning and encourages students to value their peers' opinions.

Actual interpersonal relationships between an instructor and her students are often more complex than the three forms described here. In Sections 3.2.4 and 3.2.5, I will discuss how subtle linguistic choices and the manner of a critique delivery can influence the instructor-student relationship.

### **3.2.3. Communication Modalities**

Critiquing in design studios involves a wide range of communication modalities including speech, written comment, drawing, and gesture.

#### ***3.2.3.1. Speech***

Speech is the primary communication modality used in all critiquing settings. For example, in the desk crit that Schön observed (Schön 1985), the instructor (Quist) states how to resolve the difficulties the student (Petra) faces, or what is promising or problematic in her design. Anthony (1991) notes that speech is often accompanied by other modalities such as drawing, because the instructor can deliver implicit meaning by drawing quick sketches (Ulusoy 1999). For example, architectural drawings can deliver various design ideas such as forms of spaces, location relationships among spaces, physical connections and adjacencies.

#### ***3.2.3.2. Written Comments***

Some instructors make quick notes to accompany their sketches in desk-crits. Jury members are often asked to provide students with written feedback after a final review. The advantage of a written critique is that the student need not remember everything the instructor says; the written form serves as a permanent reminder of the critique. However, except for brief notes and labels that accompany drawings, written comments are rarely used in the design studio (Bailey 2004).

#### ***3.2.3.3. Drawing: Graphic Annotation and Image***

Instructors often draw during a design studio critique as they talk and demonstrate ideas (Schön 1985). Drawing ranges from abstract diagrams to representational forms. As the instructor discovers the relationship between architectural elements presented in the student's drawings (Ulusoy 1999), she may make a simple diagram to illustrate the relationship. To suggest other building forms, the instructor may place tracing paper over the student's drawing and occasionally the instructor will draw directly on the student's drawing. The instructor may also make quick sketches on a pad of paper to show how the design might proceed or might have been done differently.

#### ***3.2.3.4. Gesture***

Instructors also gesture to indicate spatial qualities that cannot be described in drawings. Gestures can be considered "invisible" drawings, that is, drawing actions that do not leave a permanent mark. For example, the instructor may indicate the contour of a landscape with hand gestures, or point to a part of a drawing or a model while referring to it in the discussion.

#### 3.2.3.5. *Combining Modalities*

Critiquing communication seldom occurs in a single mode. Rather, instructors frequently use verbal and visual modes simultaneously to communicate their ideas to students. Multiple critiquing modalities work together and help students to understand the instructor's intentions. Bailey (2004) characterizes a good instructor as a critic who can make clear the relationship between verbal comments and visual representations.

#### 3.2.4. **Delivery Types**

The language used by the instructor is essential to the success of the critiquing session. As a critiquing session takes the form of a conversation between instructor and student, the instructor must select appropriate content, including the choice of examples and the level of abstraction, in order to maximize the student's learning.

Instructors use two 'response styles' while critiquing: *facilitative* and *directive*, according to Bailey (2004), Goldschmidt (2003), and Uluoglu (2000). This distinction follows a study of instructor's responses in English writing education by Straub (1996). The teacher's choice of response styles makes a difference to the students' subsequent actions and learning, so it is important for teacher to be aware of different ways to communicate the same content. However, the design education literature does not address this distinction.

A facilitative critique encourages a student to elaborate on reasoning and design decisions. While pointing at a specific part of a student's design a teacher may ask "*Why did you place your gallery here? Why did you make this opening here?*" Here the teacher helps the student reflect on the work,

discover design problems, and articulate design rationale. In contrast, a directive critique involves direct comments from the instructor rather than a series of questions. This style reflects the teacher's judgment. For example, Schön observes the teacher Quist saying to his student Petra, *"It's a general pass through that anyone has the liberty to pass through, but it is not a corridor. It marks a level difference from here to here – It needs to have steps or a ramp."* Here, instead of reflecting on current problems, the student would focus on future solutions.

Uluoglu (2000) and Bailey (2004) both identify five delivery types that instructors use in critiquing: **evaluation** (positive or negative) comments on the student's work, **interpretation** of the students work, **demonstration** of potential solutions or other design solutions, **introduction/reminder** of issues or strategies, and description of existing design **examples** or analogies. These are the five delivery types used in the Furniture Design Critic described in this dissertation.

### 3.2.5. Delivery

Critiquing delivery is different from the delivery types and communication modalities discussed in the previous section. Delivery is the *manner* in which instructors express comments based on their linguistic and modality choices. An instructor's body language—facial expressions and hand gestures—communicate subtle nuances of the instructor's attitudes or decisions. The voice quality<sup>5</sup> of the instructor (e.g., such as intonation and loudness) as well as manner of drawing (e.g., deliberate and slow vs. rough rapid) are also part of delivery (Roach, Stibbard et al. 1998).

---

<sup>5</sup> Voice qualities are often described in terms of prosodic and/or paralinguistic features. As these linguistic terms are not well defined, I use the term "voice quality" in this work (Roach 1998).

Anthony (1991) reports that these nonverbal aspects of communication are as important as the instructor's linguistic choices. She postulates that the effective use of nonverbal expression can deepen communication between an instructor and a student. She presents four distinct messages that nonverbal body language can convey: confirming-repeating, denying-refusing, strengthening-emphasizing, and regulating-controlling. She argues that an instructor must learn what nonverbal expressions convey to the students.

Anthony also suggests that the instructor should try to be consistent in verbal expressions and nonverbal behaviors. When an instructor's verbal expression implies a positive connotation, body language and facial expression should also communicate positive attitude. Otherwise, the inconsistency can confuse students or cause them to question what their instructor really thinks about their work. If an instructor tells a student, *"You are doing well in this part of floor plan, it looks interesting,"* while frowning at the student, the student may wonder whether the instructor is actually entertaining a different thought than what the words communicate.

### **3.2.6. Critiquing Phases**

Critiques that students receive are different, depending on the phase of design they are engaged in. Uluoglu (2000) notes that instructors decide the purpose and content of a critique according to design phase. She examined the syllabi of second-year studios at several US architecture schools (U.C. Berkeley, MIT, CMU, Harvard, and Istanbul Technical University). She identified a common six phase outline:

- (1) introduction— introducing studio goals and requirements,
- (2) place/space – investigating fundamental knowledge (e.g., site analysis),
- (3) settlement/building – early stage designing and sketching design ideas to communicate,
- (4) building (life/space) – designing by considering a building program, or concepts,
- (5) supporting knowledge – studying existing buildings and design theories, and
- (6) building (systems) – considering knowledge on building systems and details.

An instructor may help a student locate and form a building in a given site by asking questions and introducing alternative approaches in the third phase. In contrast, during the fifth phase the instructor may offer relevant precedents to lead the student to look at other architects' work with similar concepts or situations.

### **3.2.7. Individual Differences**

Although usually all students in a studio have completed a common set of required courses, individual students bring unique qualities to the learning experience, including learning style, gender, and cultural background.

#### ***3.2.7.1. Learning Style***

Education researcher David A. Kolb developed a model of learning called Experiential Learning Theory (1984). The core of Kolb's model is a simple description of the learning cycle: how experience is translated into concepts, which in turn serve as guides in choosing new experiences. The learning cycle has four stages: beginning with concrete experience (CE), advancing to reflective observation (RO), further abstract conceptualization (AC), and culminating in active

experimentation (AE). Kolb's model classifies learners into one of four learning styles by identifying characteristics (Table 3.2): convergers (AC + AE), divergers (CE + RO), assimilators (AC + RO), and accommodators (CE + AE).

**Table 3.2 Learning Styles and Disciplinary Difference**

<b>Learning Styles</b>	<b>Disciplinary Difference</b>
<b>Convergers (AC + AE)</b>	<ul style="list-style-type: none"> <li>• Practical application of ideas</li> <li>• excel in those situations, like conventional intelligence tests, where there is a single correct answer or solution to a question or problem</li> <li>• prefer to deal with things rather than people</li> <li>• <b>Physical sciences/ engineers</b></li> </ul>
<b>Divergers (CE + RO)</b>	<ul style="list-style-type: none"> <li>• Imaginative ability</li> <li>• excel in the ability to view concrete situations from many perspectives</li> <li>• perform better in situations that call for generation of ideas, such as "brainstorming" sessions</li> <li>• <b>Humanities and liberal arts backgrounds</b></li> </ul>
<b>Assimilators (AC + RO)</b>	<ul style="list-style-type: none"> <li>• Create theoretical models</li> <li>• excel in inductive reasoning, in assimilating disparate observations into an integrated explanation</li> <li>• less interested in people, more concerned with abstract concepts, but less concerned with the practical use of theories</li> <li>• <b>Basic science and mathematics</b></li> </ul>
<b>Accommodators (CE + AE)</b>	<ul style="list-style-type: none"> <li>• Doing things</li> <li>• in carrying out plans and experiments and becoming involved in new experiences</li> <li>• excel in situations that call for adaptation to specific immediate circumstances</li> <li>• tend to solve problems in an intuitive trial-and-error manner, relying heavily on other people for information rather than their own analytical ability</li> <li>• <b>Technical or practical fields</b></li> </ul>

### **3.2.7.2. Spatial Ability**

In architectural design both instructor and students often use visual representations: sketches, 3D computer graphics, and physical models. Therefore, each student's spatial ability is a factor that influences learning. According to cognitive scientist Richard Mayer (2001), learning materials that

combine text and graphics (as opposed to text only) work better for learners who lack prior knowledge about the subject matter and for learners who have high spatial ability.

#### ***3.2.7.3. Gender, Race, and Culture***

Some researchers argue that design studios should value diversity such as the students' gender, race, cultural background, and ideologies (Willenbrock 1991; Koch, Schwennsen et al. 2002). Based on personal experiences in an undergraduate architectural program, Willenbrock (1991) suggests that more equal exchange between instructor and students as well as among peers, is needed in design studio.

Several researchers point out that students' different identities (e.g., gender, race, experience, and cultural background) may influence learning and dialogue with instructors. Boyer and Mitgang (1996) note that (in the USA) architecture remains an undiversified professional domain dominated by white males, according to the statistical data on female and minority students and faculty in accredited architecture programs. Some researchers report that the master-apprentice model reinforces the image of men as "masters" (Ahrentzen and Anthony 1993). Most design studios follow the master-apprentice model and most architecture studio instructors are male, which frequently makes design studio teaching patriarchal (Willenbrock 1991). Consequently, these researchers express concern that critiquing in a patriarchal learning climate can disempower and discourage some students. Some students experience emotional difficulty and damage to self-esteem when subjected to fierce public criticism (Anthony 1991; Ahrentzen and Anthony 1993).

### **3.2.8. Students' Knowledge and Experience**

Although I did not find any studies that attempt to establish the relationship between critiquing and a student's level of experience or knowledge, it is reasonable to assume that an instructor would consider what a student does (not) already know in critiquing sessions. In particular, it would be useful to understand what design experience and knowledge can be expected from a student based on their stage in the educational program.

Some studies compare the design processes employed by beginning design students and upper-level students. For example, Atman et al. (1999) analyzed verbal protocols of 24 seniors and 26 freshmen given an open-ended design problem and compared their design processes. They found that the design processes of seniors and freshmen differ in several dimensions: scoping problems, considering multiple alternatives, transitions between design stages, and paying adequate attention to each design stage. Compared to beginning design students, experienced students considered multiple alternatives and could transition more rapidly between design stages while paying closer attention to each stage. This and other studies suggest that in order to provide effective feedback it is important for the instructor to understand a student's level of experience and design ability.

### **3.2.9. Students' Response Types**

Critiques require students to reflect on their instructors' comments. Some students may grasp the feedback they receive; however, others have difficulty relating to the feedback. Rohrbach (2005) suggests that design teachers lack understanding about how to reconstruct their teaching methods, specifically critiquing, according to students' responses. It is reasonable to assume that teachers consider students' response types in order to reconstruct their critiquing methods. Through

interviews after critiquing sessions Kent (2001) identifies six types of students: thinkers, listeners, skeptics, followers, misinterpreters, and the affirmed. Thinkers reflect on what they hear and they have ideas about how to incorporate the feedback into their designs. Listeners hear and can repeat the teacher's opinions; however, they cannot come up with a clear plan to act. Skeptics are cynical or skeptical about the teacher's comments and tend to discount the feedback they receive. Followers remember and use the teacher's concrete comments. Misinterpreters misunderstand the teacher's feedback and turn it into what they want to hear. Finally, the affirmed feel that their teacher agrees with their ideas.

### **3.2.10. Design Artifacts and Learning Goals**

Two additional factors have not been discussed. The first is the variety of artifacts that students produce as they develop their designs. Design artifacts include diagrams, rough sketches, drawings, and physical models. Critiquing is primarily offered based on those artifacts, so it is important for the instructor to understand the student's ability to produce them. The second factor is the learning goals that the instructor must address. Regardless whether they are articulated explicitly, each studio course has a set of learning goals. In order to provide students with appropriate feedback the instructor must clearly understand these learning goals.

### **3.3. Summary: A Framework of Critiquing Practice**

The literature of architecture and design education suggests a process model of critiquing and eleven fundamental factors that make up critiques. The factors are divided into two groups: methods and conditions. Depending on various conditions—the phase of the designing, the knowledge and

expertise of the student, and so on, a critic can select from a variety of methods to offer a critique—for example, choosing a desk crit or group crit setting, using speech or drawing, whether to play the role of master, user, or peer.

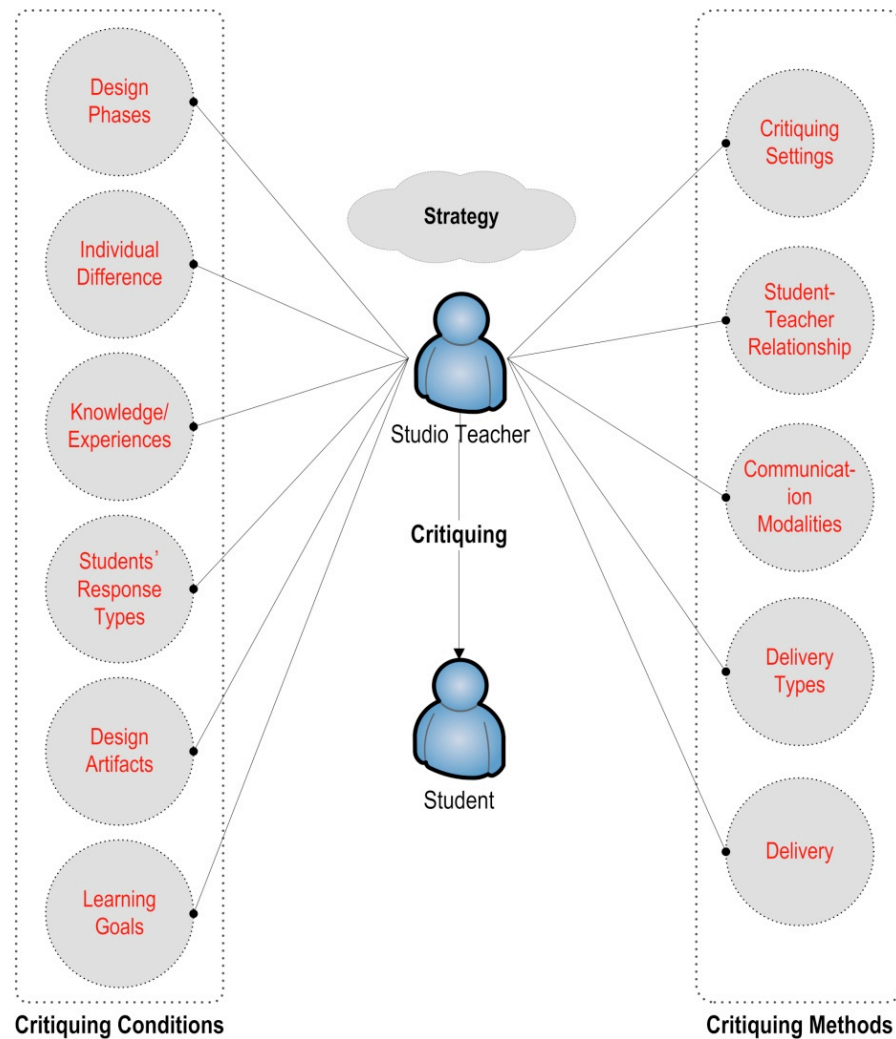
**Table 3.3 Key Factors of Design Critiquing**

	<b>Factors</b>	<b>Details</b>
<b>Critiquing Methods</b>	Settings	Desk crit, Group crit, Interim review, Final review, Informal interaction
	Communication Modalities	Speech, Drawing, Text, Gesture
	Student-Teacher Relationship	Master & Apprentice, User & Designer, Peer-to-peer
	Delivery Types	Facilitative, Directive - Introduction/ reminder, Interpretation, Example, Demonstration, Evaluation
	Delivery	Manner in which instructors express comments based on their linguistic and modality choices, e.g., voice quality, facial expression, etc.
<b>Critiquing Conditions</b>	Design Phases	Structures of design studios
	Individual Differences	Student's learning styles, spatial ability, gender, race, & cultural background
	Knowledge/Experiences	Student's level of knowledge and experiences
	Students' Response Types	Thinkers, Listeners, Skeptics, Followers, Misinterpreters, & the Affirmed
	Design Artifacts	Drawings (Diagrams, Rough sketches, etc.), Physical models
	Learning Goals	Learning goals that a studio instructor addresses in studios

Table 3.3 summarizes these eleven factors and their characteristics. I have described each factor and discussed relevant educational implications. Figure 3.3 illustrates how a critic (whether a human teacher in a design studio, or a computer-based critic) can use these factors in deciding on a strategy to critique a student.

This framework might be useful in developing a more formal and rigorous pedagogy for design education. As mentioned above, although critiquing is the backbone of studio-based education, design educators do not learn critiquing formally, nor has critiquing been the subject of learning sciences research that could result in more effective teaching practice.

More relevant to the project described in the later chapters of this dissertation, this framework of critiquing practice provides a foundation to develop the mechanisms of a computer-based critiquing system. It outlines the critiquing conditions that a system can recognize, and the critiquing methods a system can use. Ideally, a computer based critiquing system would incorporate all these factors. However, that is beyond the scope of this work.



**Figure 3.3 A Framework for Critiquing Practice:** Conditions and Methods. A studio teacher considers critiquing conditions and then selects a set of critiquing methods to offer feedback.

The Furniture Design Critic program described here is restricted to a subset of this model. The program functions only in one particular critiquing setting: the desk crit, and it focuses on only two of the critiquing methods that the model identifies: delivery type and communication modality. It remains for future researchers to build more inclusive computer based critiquing software that can

take into account more of the critiquing conditions in this framework, and employ more of its critiquing methods.

## 4. Intelligent Tutoring Systems

An intelligent tutoring system (ITS) is an educational program that tracks a student's actions, and generates feedback. It supports learning by detecting and solving problems. It is similar to a critiquing system in that it monitors users' actions and offers feedback to help them solve problems (Robbins 1998). Intelligent tutoring systems provide rich context-sensitive pedagogical assistance according to student models that the systems construct to represent the individual student's knowledge level.

A study of Intelligent Tutoring Systems suggests ways to implement a program to realize the critiquing model presented in Chapter Three. In particular, intelligent tutoring systems represent domain knowledge and make inferences about a user in order to generate individualized feedback. This chapter reviews the system architecture of intelligent tutoring systems, compares two main approaches to building these systems, and describes why one of these approaches, constraint-based tutors are a more appropriate architecture for the Furniture Design Critic.

## 4.1. ITS System Architecture

A typical intelligent tutoring system consists of four main components: *domain model*, *student model*, *pedagogical module*, and *interface* (Heffernan, Koedinger et al. 2008). Figure 4.1 shows the flow of information among these components.

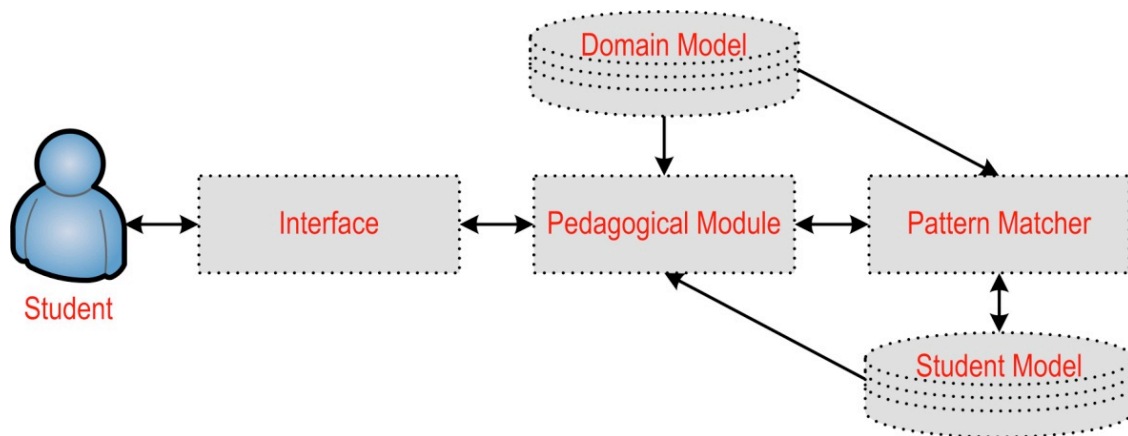


Figure 4.1 System Architecture of an Intelligent Tutoring System

The *Domain Model* contains knowledge expressed as facts and rules about the domain that the system uses to check a student's actions. This knowledge ranges from extensive, complete knowledge required to solve problems to a subset of domain knowledge required for teaching. Some intelligent tutoring systems can generate the correct solution according to the student's problem-solving path. The PACT Algebra tutor (Corbett, Trask et al. 1998) is one such system. It generates the correct solution and the solution path based on the student's actions. Other tutors represent only the domain knowledge needed to validate a student's actions. For example, SQL Tutor

(Mitrovic 1997) checks and provides feedback on a solution proposed by a student, but it cannot generate solutions.

The *Student Model* (corresponding to the User Model in a computer-based critiquing system) represents the state of a student's knowledge and skills. A tutoring system dynamically maintains its Student Model by evaluating the student's actions, and infers the student's knowledge by comparing the student's solutions against the tutor's solution. The tutor's pedagogical decisions depend on the Student Model. If the tutor makes faulty inferences about a student, then it may make poor decisions for individualizing feedback and selecting the next problem. Researchers in intelligent tutoring have developed techniques for maintaining both short-term and long-term student models. The two main methods used to build the short-term model are model tracing (Anderson, Corbett et al. 1995) and constraint-based modeling (Ohlsson 1994); these are discussed below. The main methods used to build long-term student model are overlays (Holt, Dubs et al. 1994) and stereotypes (Rich 1989). Overlay models represent a student's knowledge as a subset of the domain knowledge. In the initial state the model assumes that the student knows nothing and the system constructs the long-term student model it interacts with the student. A stereotype model classifies the student into levels of knowledge. This model often starts with a pre-test score.

The *Pedagogical Module* plays an important role in tutoring systems. It takes information from the Student Model and the Domain Model and decides what to present, when to present it, and what problem to give next to the student. To support learning, this Pedagogical Module can adopt different methods and strategies. It can decide to enforce correctness or lead the student to construct knowledge by posing questions or using self-explanation.

The *Interface* is where the tutor presents the student with problems to solve, and then offers feedback to the student. The student makes external representations to solve the task at hand, and the tutor presents feedback to guide the student toward task completion. For example, the Interface in the PACT Algebra tutor provides a workspace where a student can enter the equations for a given algebra problem. This tutor then annotates these equations with corrections or suggestions.

## 4.2. Two Main Approaches of Intelligent Tutoring Systems

Two main approaches are used to implement intelligent tutoring systems: model-tracing, or cognitive, tutors, and constraint-based tutors.

### 4.2.1. Model Tracing Tutors

Model tracing, sometimes called ‘cognitive’ tutors are regarded as the most successful in the field. Model-tracing is based on the ACT-R theory of Anderson, Corbett, et al. (1995). The central idea is that people use two memory modules: a declarative module and a procedural module. In order to perform a task, a person needs both the relevant declarative and procedural knowledge. Declarative knowledge consists of facts that the student uses, for example mathematical theorems (e.g., the sum of angles in a triangle equals 180) or architecture spatial zoning rules (e.g., the kitchen and the dining room must be adjacent). Procedural knowledge is the knowledge a student needs to accomplish a task, such as how to apply the triangle theorem in a mathematical proof or the spatial zoning rule in an architectural design. ACT-R represents this knowledge as production rules. Learning is described

as the following steps: students must acquire declarative knowledge, and then they must transform it into procedural knowledge.

A model-tracing or cognitive tutor can follow a student who is performing a task correctly by matching the student's sequence of steps with a set of production rules that describe the domain model. This model maps all the possible paths a student may follow in order to solve a specific problem. As a student solves a problem, the tutor traces the student's cognitive model by checking that each step matches a rule. When the tutor cannot find a rule that matches the student's move, it indicates that the student has made an error.

Production rules are extremely important in these systems. Tutoring is seen as the process of transferring production rules needed to solve problems from the system to the student. For example, consider a set of production rules for calculating the third angle of a triangle using two angles already known. The theorem of three angles of a triangle is that the sum of three angles is  $180^\circ$ . This set of production rules consists of two if-then rules: (1) checks if the current goal is to find all three angles of a triangle and if two angles are already known; (2) if the first condition is satisfied, the tutor computes the unknown angle. Because production rules are procedural in nature as the above example has showed, it takes a long time to identify and write the chains among if-then rules. Composing the set of production rules for a cognitive tutor is a labor-intensive and time-consuming process. Anderson and his colleagues estimated the time to identify and design a single production rule was ten hours or more (Anderson, Corbett et al. 1995). Koedinger and his colleagues (2004) estimated that an average of 200 hours of development time was required to produce one hour of educational content.

Model tracing (cognitive) tutors have been developed for diverse domains such as algebra (e.g., PACT Algebra tutor (Corbett, Trask et al. 1998)), geometry (e.g., PACT Geometry tutor (Aleven and Koedinger 2000)), and programming (e.g., Lisp Tutor (Anderson, Farrell et al. 1984)). These tutoring systems have been coupled with knowledge tracing technique as a long-term student modeling method. Knowledge tracing technique tracks a student's overall learning of target knowledge in the tutor system. The tutor models student's understanding as a collection of knowledge components. These components are assumed to be either '*known*' or '*unknown*'. The tutor estimates the probability that each target knowledge component is known. This enables the tutor to identify students' individual difficulties and to present problems targeting specific skills that a student has not yet mastered.

#### **4.2.2. Constraint-based Tutors**

Constraint-based modeling is a method for domain and student modeling proposed by Ohlsson, based on his theory of *learning from performance errors* (Ohlsson 1996). According to this theory, learning occurs when we catch mistakes or when others catch them for us. We make mistakes because we have not mastered the declarative knowledge need, or cannot apply it in the task conditions, or because there are too many decisions for us to make while performing the task. In order to solve problems correctly, in addition to learning declarative knowledge we must learn to apply it in particular situations.

Constraint-based tutors have been developed in domains such as database design (e.g., SQL-Tutor (Mitrovic 1997), KERMIT (Suraweera and Mitrovic 2002), ERM Tutor (Milik, Marshall

et al. 2006), NORMIT (Mitrovic 2002)), punctuation (e.g., CAPIT (Mayo, Mitrovic et al. 2000)), and language (e.g., WETAS (Martin and Mitrovic 2002)).

The key point of constraint-based modeling is that the diagnostic information is in the problem state rather than the path taken to arrive there. In other words, all valid solutions satisfy the general principles of the domain. Domain knowledge is modeled using constraints to represent the application of a piece of declarative knowledge to a particular situation. Each constraint represents an item of domain knowledge; it consists of a relevance condition and a satisfaction condition. The relevance condition indicates when the constraint applies—and the satisfaction condition represents states where a certain piece of knowledge has been applied correctly. Therefore, a solution must satisfy the satisfaction condition, when the constraint is deemed relevant to the user's solution. A violated constraint indicates an opportunity to improve the proposed design. The tutor then offers feedback regarding the violated constraint.

A constraint-based tutor records information about a student to deliver individually tailored instruction. This *student model* consists of the history of all constraints that the tutor has applied to the student's design, including both satisfied and violated constraints. The violated constraints indicate domain knowledge the student has not yet mastered. Based on this diagnosis, the constraint-based tutor provides feedback to help the student improve the solution.

#### **4.2.3. Constraint-based Tutors for Design Critiquing**

I adopted the constraint-based modeling (CBM) approach to developing the Furniture Design Critic system, because this approach seems suited for design domains. There are several reasons for this

decision. First, CBM does not require an extensive and complete domain model or an expert module that can generate correct solutions, whereas model tracing does. However, it is difficult or impossible to formulate all possible actions and solutions for design problems. For example, design programs lack clear distinctions between right and wrong solutions. Instead there often are competing reasonable answers. It is also impossible to identify all possible actions and problem-solving paths. Constraint-based tutors can generate pedagogical assistance without being able to solve problems, by focusing on violated constraints.

Second, model-tracing tutors require a model of all the desirable paths from problem to solution. Without this, the tutor would be unable to trace the student's actions. In contrast, the effect of a missing constraint is highly restricted in constraint-based tutors, because each constraint provides only a partial description of a solution. A missing constraint may result in failing to identify a particular opportunity to improve a solution but the solution can still be validated and analyzed against the other constraints, because the constraints are modular in nature. This enables us to build the domain model incrementally. For example, the SQL tutor has been implemented, used, and improved, although the model is still incomplete. This approach is more appropriate for supporting design learning because it is unrealistic to attempt a complete domain model for design problems.

Finally, model-tracing tutors are inherently restrictive, because their production rules are procedural in nature. Although these tutors allow students to deviate from a correct solution path, this is practically limited by the need to determine when the student has done something wrong and to understand what they have done wrong. In contrast, constraint-based modeling only focuses on

the particular state that a student arrives at. Therefore, students are free to take whatever approach they please, and to change strategies in the midst of problem solving.

### **4.3. Summary**

This chapter outlined the typical system architecture of intelligent tutoring systems and briefly explained the main components of these systems such as Student Model and Pedagogical Module. The two main approaches to developing domain models and student models are model tracing and constraint-based modeling. I compared the two approaches and explained why I adopted the constraint-based approach for the Furniture Design Critic.

## 5. The Furniture Design Critic

### 5.1. The Program: Why a Computational Model?

The literature review of design studio education (in Chapter 3) revealed that our understanding of design critiquing is at best incomplete. Design educators and those studying design education have not articulated how to conduct design critiques, nor have we empirical data about how studio critics decide on critiquing strategies.

Based on the framework for critiquing practice presented in Chapter 3, this chapter proposes a computational model of design critiquing, which is implemented in the Furniture Design Critic program. The program provides an experimenter with a vehicle for representing and manipulating critiquing conditions and critiquing methods, delivery types and communication modalities. This model demonstrates plausible ways to decide on critiquing methods in the consideration of critiquing conditions. Thus the Furniture Design Critic program is a tool for modeling design critiquing.

I selected flat-pack furniture design domain as a test domain for this computer-based critiquing program. Flat-pack furniture making is an interesting design domain where students encounter many structural and spatial design issues as they make stable furniture out of the flat materials. Compared to architectural design the problem space of flat pack furniture design is relatively small, so we avoid the need to account for the large body of domain knowledge that underlies architectural design. Still, like architectural design, flat pack furniture design is an ill-defined and open-ended domain. Consequently, the Furniture Design Critic employs critiquing methods that are also used in architectural education.

The Furniture Design Critic program selects a particular set of critiquing methods from five delivery types and three communication modalities. I briefly described these critiquing methods in Sections 3.2.3 and 3.2.4; here I closely examine these methods using detailed examples.

This chapter illustrates the Furniture Design Critic's decision-making process using three examples. This chapter describes what characteristics of a certain designers the program infers and how and why it selects particular sets of critiquing methods. (The implementation details of the Furniture Design Critic program are described in the following chapters.) Three example sessions are described in Section 5.5 using these selection mechanisms of the critiquing methods to show the critiquing model.

## **5.2. The domain: Flat-pack Furniture Design**

Flat-pack furniture design is fun and easy, which is why design schools often use it as the first exercise for first-year students. Students become familiar with design problem-solving by drawing

and modeling. They are given a project with a small set of design constraints. They learn to satisfy basic functionality and pre-determined criteria through a series of three-dimensional experimental compositions of planar wooden elements.

Flat-pack furniture making is used as the first studio exercise in the School of Design at Carnegie Mellon University. In this studio, students are asked to generate three-dimensional solid wooden forms that demonstrate a structure of planar parts at right angles in a visually balanced composition. The final project of this studio is to design wooden stools. Their final design must hold a 200 lb person above the ground. Students are asked to utilize the inherent qualities of the materials (planar wood) to create a balanced form and to address issues of structure, support, and stability. Students submit physical models and drawings to explain their design ideas and to show how the flat elements are constructed and jointed together.



**Figure 5.1 Final design submissions of several students for their Flat-pack Furniture Making Exercises:** (a) Gregory Zulkie; (b) David Rocco; (c) Mary Katica

Figure 5.1 shows the designs that three students submitted at the conclusion of the design studio (Fall 2005 and Fall 2008). Each student generated a series of drawings and scale models as well as their final work in full scale. Figure 5.2 shows a final prototype with plan, section, elevation, and axonometric drawings.

While designing, students are concerned about making their chairs stable, which structure is appropriate, how to support structurally important parts, the function of each part, and how to assemble the parts. They also make drawings and scale models in order to reflect on their design process and to check their chair designs.

As students design flat-pack furniture by making drawings and scale models, they gradually acquire design skills, and they learn to handle design problems with given constraints. Students also gradually gain the ability to use visual design representations such as drawings and scale models to develop their design solutions.

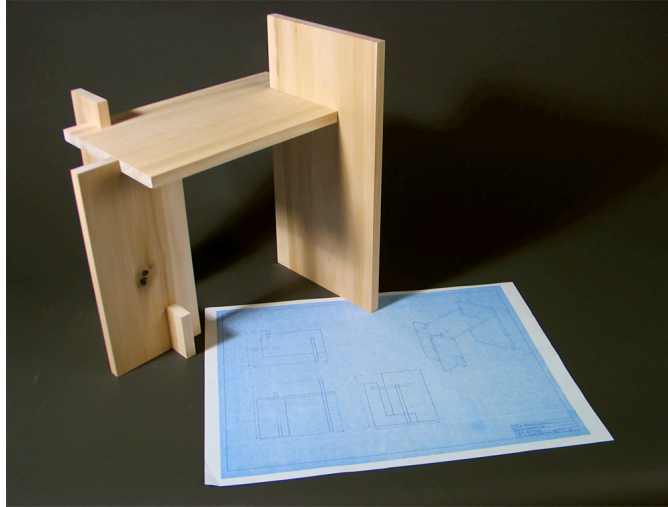


Figure 5.2 Wooden stool and drawing including a plan, elevation, section, and an axonometric view (designer: Lee Byron)

### 5.3. The “5x3” critiquing model: Delivery Types and Communication Modalities

The Furniture Design Critic program offers feedback using five delivery types and three communication modalities. Table 5.1 shows the space composed of these methods that the program uses. Delivery types cannot exist alone, nor can communication modalities. Together these delivery types and communication modalities define critiquing comments. I have identified this two dimensional space, because it helps describe a wide range of combinations with delivery types and communication modalities. The horizontal axis, facilitative – directive, features the five delivery types. They include (1) *interpreting* designers’ solution, (2) *introducing* design ideas or approaches/*reminders* of them, (3) description of existing *examples* or precedents, (4) *demonstrating* potential solutions or other design actions, and (5) *evaluating* (positive or negative) of the designers’

solutions. The vertical axis includes three communication modalities: written comments, graphical annotations, and images.

**Table 5.1 Space of Delivery Types and Communication Modalities**

<div> <div>Delivery Types</div> <div>Communication Modalities</div> </div>	Facilitative → Directive				
	Interpretation	Introduction/ Reminder	Example	Demonstration	Evaluation
Written Comments					
Graphical Annotations					
Images					

When the program identifies critiquing opportunities, it offers feedback in various forms, choosing from among the critiquing methods. For each critiquing opportunity the program stores five different written comments. Table 5.2 illustrates this with two examples for each delivery type. The choice of delivery types is important because it may influence a designer’s subsequent actions and hence reflection. For example, when a critic offers an example, a designer may realize that reference to the given precedent is helpful to deal with the new design situations and attempt to adapt it to fit the work at hand. When the critic points out errors, the designer may quickly see the problems and fix them. Thus the use of different delivery types in critiquing may lead to different reasoning.

**Table 5.2 Feedback Instances of Five Delivery Types**

<b>Delivery Types</b>	<b>Instances of Written Comments</b>
<b>Interpretation</b>	<ul style="list-style-type: none"> <li>• Your chair has three legs. They support loads at the center. Maybe you mean to support the weight of a user sitting at the center.</li> <li>• Your bookcase has a top, a shelf, and two sides. The important structural parts to support loads are only two sides.</li> </ul>
<b>Introduction/Reminder</b>	<ul style="list-style-type: none"> <li>• A user cannot keep upright posture while sitting on the chair. The center of gravity can move around.</li> <li>• Do you think the back part of your bookcase is big enough to support the lateral load?</li> </ul>
<b>Example</b>	<ul style="list-style-type: none"> <li>• Look at other tables that have the same issues as your design. How you can resolve the common issues</li> <li>• Look at other chairs, see other designers make chairs differently.</li> </ul>
<b>Demonstration</b>	<ul style="list-style-type: none"> <li>• You can add legs to support the unsupported corners (drawing parts that represent legs on a designer's diagram). They will improve stability.</li> <li>• You can add a vertical part to support the shelf (drawing a vertical part between two shelves).</li> </ul>
<b>Evaluation</b>	<ul style="list-style-type: none"> <li>• Your chair has only one armrest, which is good, but it breaks symmetry on your design.</li> <li>• Braces cause discomfort, because they do not allow enough leg space. Consider removing the braces or moving them to other positions.</li> </ul>

This section introduces the space of delivery types and communication modalities that the Furniture Design Critic program uses. Also it also provided the feedback instances. The following section will look at a real critiquing session to see how these introduced critiquing methods are used.

#### **5.4. An Example from Real Life: Analysis of an Desk Crit Session**

I analyzed an undergraduate architecture studio desk crit published via Open Courseware at Massachusetts Institute of Technology (Wampler 2002). This session reveals critiquing methods used in a real studio setting. Here the studio teacher (Wampler) uses the delivery types and communication modalities identified in the design education literature. Wampler offers feedback in

various delivery types (question, evaluation - positive or negative, interpretation, demonstration, introduction/ reminder, and examples) and modalities (graphical annotations and images, etc.).

The task was to design a “community meeting place” for a group of people. The project involved making a small building, not more than 5,000 square feet, to contain meeting places, work areas, and residential space. Adjacent to this building would be a small natural/recreation space for the neighborhood to enjoy.

In this desk-crit, a student and a teacher discuss the student’s work. The student first briefly reminds the teacher what she presented during their last critiquing session, pointing at her old physical model. The student conveyed the notion that the teacher knows her and her design, saying, “Now you know me pretty well.” She describes what she has done to her design since their last meeting. She repeated that she didn’t want to cut down any trees. She explained what she was trying to do by describing the spaces she designed: an alley besides her building, a secret garden in the back yard, a second-floor balcony, and a patio area for dining. As she talked and pointed to her drawings and model, the teacher raised questions or rephrased what the student said in order to check his **interpretation**. For example, at one point the teacher asked, “... *which means that this space is a two-story space?*” and “*Oh, you are just getting light into this level (pointing out the bottom window in the physical model).*”

The teacher started by offering **positive comments** because he is pleased with the student’s progress since their last session:

**Teacher:** “First of all, the evolution from this (pointing to her previous physical model) to this (pointing to her current physical model) is very successful; because I think you are starting to

develop architectural language in this (pointing her current physical model), which obviously needs more work. Nevertheless, you have converted the rough form into something more precise. I think this is good.”

Next, the teacher makes **negative comments** and **gives an example** about how to resolve the issue.

**Teacher:** The notion of the ramp is to send people through this and across over (here), which is a new idea. This is very powerful, because the normal ramp is one twelfth? Probably it is not that.

**Student:** It is not.

**Teacher:** It's OK. It's for the handicapped. That's not necessarily, but the notion of ramp directs me through this way, which is a very powerful notion. I have some comments. The idea of this direction still involves covered lights. I think that it's powerful. I doubt if it's necessarily continued in the same way. For instance, here, it could be outside like the walkway in our architecture building, or slightly covered, but not necessarily heated. The heated space is here. So I can come into the heated building and then pass through the uncovered walkway and come out and then I can go back into it.

The teacher **asks questions** to introduce an important idea and to let the student think about it before he points it out. The teacher asks,

**Teacher:** “If I am coming down this street for the first time, do you see me entering into this space (an entrance at the left of the building) or into here (the main entrance) as an entrance?”

**Student:** “hmm, I think this (the entrance in the left side) is more permanent. More people enter here at first, and then find out what this is. Maybe I have to place a wall here (pointing to her main), to invite the people into the building, even though this is where my quieter spaces are - for studying, sleeping or other things.

After listening to the student’s answer, the teacher explains why he asks the question and **introduces the issue**. The teacher also **suggests possible moves** by **demonstrating**, for example, setting up a small piece of cardboard to indicate a wall in front of the right entrance. He also explains that this wall can direct people toward the left entrance.

**Teacher:** “You have a very subtle but important problem in architecture, namely that you have two entrances. This (left entrance) is major and this (right entrance) is important for you, but it (right entrance) is probably not as major as this one (left entrance). It is because of the directionality you are working with.

You need to make a form here. Can I use this (a small piece of cardboard)? You need to do something with this (he sets up a wall in front of her right entrance). Not necessarily that, but something like that. This directs me into this place a little bit more forcefully. Can I have a piece, some toys (cardboard)? This is the extreme case (placing a high wall instead of the small wall). I don’t mean like this (making the wall shorter). I come to this point and I am directed into this space here. At the same time, I can go in here, but this is probably an entrance that I think will lead into

the building, if I don't know this building. I think that you need to pronounce it more here. I am leading up something here, because directionality could be better exploited in your design. However, at the same time, it seems to me that by coming into this space (right entrance) I can experience this wonderful ramp. This ramp can turn here to pick me up. I can both go in and come out with this ramp. It's a clear path of your journey as you move through this space."

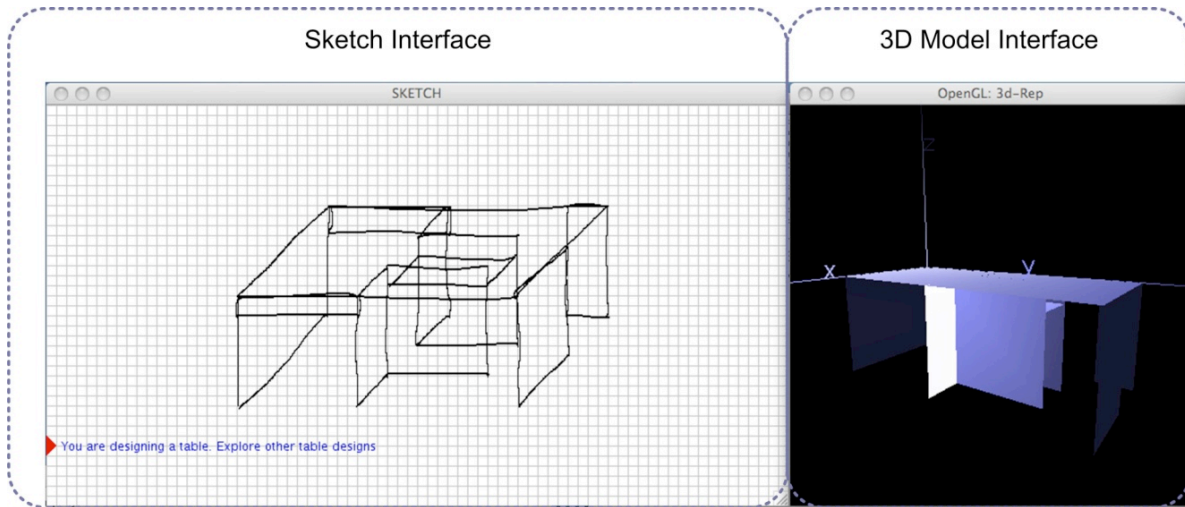
He also **demonstrates** how to develop the student's design **by sketching over her drawing**. As he draws, he first checks the building's context, such as trees and walls, with the student. The teacher makes suggestions **by sketching on her drawing**, combining two spaces by cutting a triangular shape from one mass. The teacher covers the student's drawing with transparent paper and makes **graphic annotations** on it, while presenting opinions and alternatives.

### 5.5. Furniture Design Critic: Three Vignettes

The Furniture Design Critic program is an interactive computational environment where a designer develops and receives feedback on a furniture design (Figure 5.3). The designer draws a diagram on the Sketch Interface and the program constructs a 3D model from the diagram. The program's sketch-to-3D functions are part of the Furniture Factory, a program I wrote previously to support novice designers in constructing flat-pack model furniture made using a laser cutter (Oh, Johnson et al. 2006). A simple sketch recognition program identifies planar polygons in the drawing, and based on an inferred 3D axis system, generates a 3D model. Then a set of jointing rules help the designer by specifying how the planar parts meet, and the program then generates an HPGL format file that a laser cutter uses to cut the parts out of basswood or cardboard.

The Furniture Design Critic is a computer-based critic that offers feedback to help the designer improve the design. The program presents critiques in the form of written comments and graphical annotations in the Sketch Interface window. In a separate window it also presents critiques in the form of images (Figure 5.10).

The program generates critiques by diagnosing the designer's sketched design. A set of condition-action rules, or design constraints, describe conditions that a design must satisfy. For example, one such condition asserts that a chair must have a seat; another asserts that a bookcase must sustain lateral loads. The program uses these design constraints to identify opportunities for generating a critique. Once the program has identified one or more critiquing opportunity, it must decide which critique to present first (if there are more than one) and how to present the critique to the designer. It uses a set of critiquing rules (distinct from the design constraints) to decide how to present the critique. These rules take into account various critiquing conditions, as described in the vignettes that follow.



**Figure 5.3** The Furniture Design Critic interface is composed of Sketch Interface and 3D Model Interface

This section presents three vignettes that illustrate the Furniture Design Critic program's underlying framework of critiquing and the program's use of the framework in selecting critiquing methods, delivery types and communication modalities. Each vignette presents a different short critiquing session in which the Furniture Design Critic program offers feedback to help a designer improve a design. In each, because it has been given a different set of critiquing rules and because it is presented with a designer with a different profile, it selects different critiquing methods. These vignettes introduce the Furniture Design Critic program before describing how it works. Later, Section 5.4 will explain the underlying system mechanisms; specifically, how and why the Furniture Design Critic selects a particular set of critiquing methods.

In the first vignette (Section 5.5.1) the program selects delivery types and communication modalities by considering only the designer's profile. It considers how much the designer knows

about the domain of furniture design, any weaknesses specific to the designer, and what critiquing methods the program has found to be effective. In the second vignette (Section 5.5.2), in addition to the designer's profile, the program considers what critiquing methods were previously used. The Critic stores the critiquing methods used on a certain design issue that it has identified. The program considers these stored methods for selecting critiquing methods. In the third vignette (Section 5.5.3), in addition to the designer's profile, and the critiquing methods that the program previously used, (the interaction history with individual designers), the program considers the combination between delivery type and communication modality candidates. The interaction history<sup>6</sup> contains what critiquing methods were selected and used, and whether the feedback presented using the critiquing methods worked or not throughout all critiquing sessions.

Ann, Ben, and Claire are three first-year design students with rather different profiles. Having worked with these students previously, the Furniture Design Critic understands how much each student knows about furniture design, his or her specific weaknesses, and what critiquing methods are effective to communicate with each of them. Ann doesn't know much about furniture design. The program knows that beginners respond well to evaluative feedback. Ben, on the other hand, is fairly knowledgeable. Claire's performance has not been excellent and she is especially weak in structural knowledge—the ability to make stable furniture designs.

In each vignette, as Ann, Ben, and Claire develop their designs, the program first considers the critiquing conditions, then selects critiquing methods, and finally offers feedback using the

---

<sup>6</sup> This interaction history contains all '*states*.' A state represents critiquing situations: (1) the selected set of critiquing methods and (2) whether the selected critiquing methods were effective (See Chapter 6).

selected methods. Each designer has a different profile, so in each case the Furniture Design Critic chooses different critiquing methods.

#### **5.5.1. Vignette 1: Ann, a beginner**

In the first vignette the Furniture Design Critic decides which critiquing methods to use, considering only how much the designer knows, the designer's specific strengths and weaknesses, and the methods that have previously been effective with this designer. Ann, the designer in this vignette, is a beginning design student.

Ann starts by sketching a bookcase. The Furniture Design Critic looks at her sketch and interprets her design. Analyzing the parts of the furniture and the relationships among them, the Critic notices opportunities for Ann to improve her design. The Critic notices three issues:

- (1) Ann's bookcase has a shelf that will bend when it is loaded with books; and
- (2) her design cannot sustain lateral loads; and
- (3) without a back the bookcase will allow books to fall behind the bookcase.

Now that it has identified some problems (or "critiquing opportunities") the Furniture Design Critic plans which issue to address first. The Critic decides to address the bending shelf issue first, because this issue is more critical than either the lateral stability issue (though this is also serious) or the problem of falling books.

In order to select appropriate critiquing methods the Furniture Design Critic considers Ann's profile. It knows that Ann is a novice designer and that the most effective delivery type for critiquing a novice is evaluation, communicated through comments and markup on the design. Therefore the Critic selects the 'evaluation' delivery type and the communication modalities 'written comments' + 'graphical annotation' (Figure 5.4). It writes the evaluative comment, "*The shelf of your bookcase is too long.*" As part of the evaluation the Critic also points out why a long shelf is problematic: "*The shelf will sag.*" It also annotates Ann's diagram, highlighting the long shelf to indicate the relevant part and drawing arrows to indicate the vertical load. The feedback leads Ann to see the problem in her design. The 'graphical annotations' on Ann's diagram help her understand the 'written comments'.

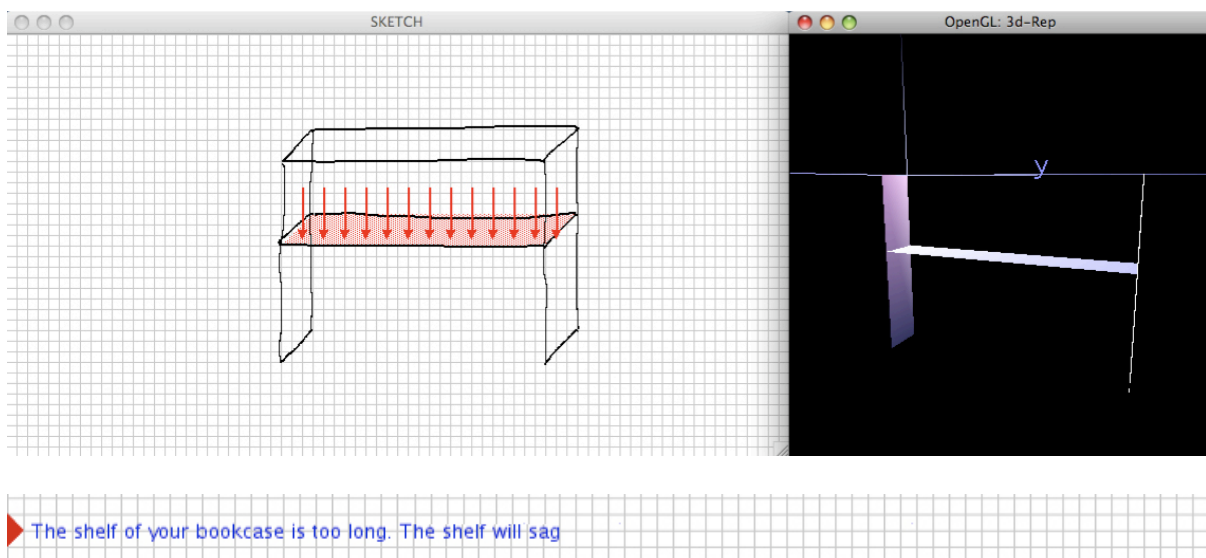
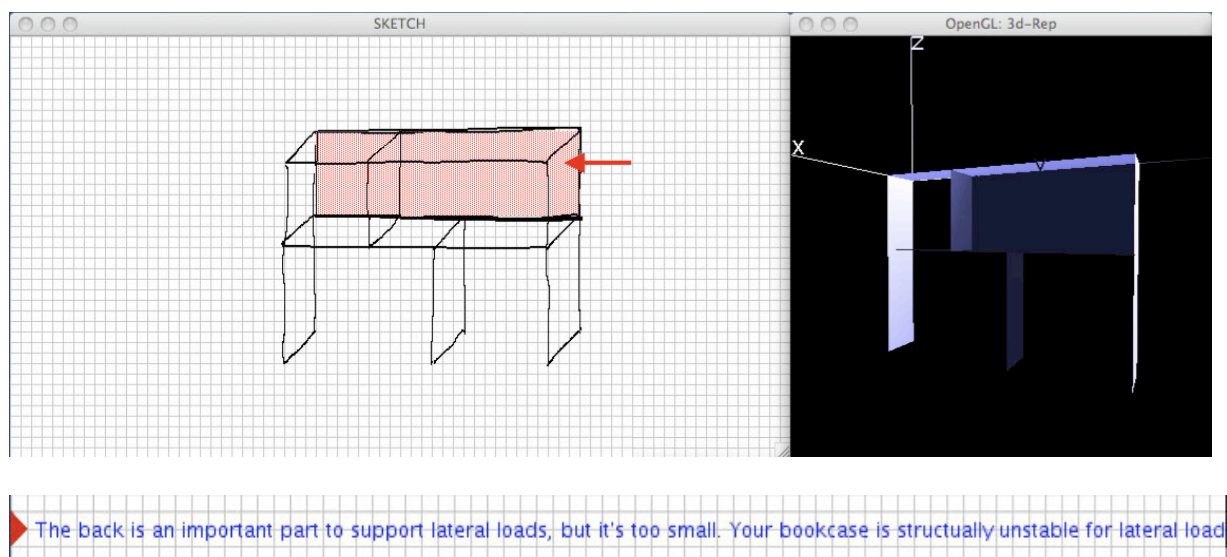


Figure 5.4 The Furniture Design Critic observes Ann's bookcase design. The Critic points out a problem and explains why it must be fixed. Also the Critic annotates Ann's diagram. The red arrows indicate the vertical loads

Ann adds a vertical support to resolve the problem; it divides the long shelf span. She also adds a back panel to prevent books from falling behind the bookcase. The Furniture Design Critic observes the changes. Ann fixed the sagging shelf problem, but the lateral stability problem remains. The Critic is only programmed to consider Ann's level of expertise, which still indicates that Ann is a novice. So again the Critic offers critiques in the same form as before (Figure 5.4): the 'evaluation' delivery type and the communication modalities 'written comments' + 'graphical annotation' (Figure 5.5). It writes the feedback in the form of evaluation *"The back is an important part to support lateral loads, but it's too small."* Again, as part of the evaluation, the Critic also describes why the small back part is problematic, *"Your bookcase is structurally unstable for lateral loads."* And the Critic annotates Ann's diagram. It highlights the back and draws a red arrow to indicate lateral loads on the bookcase to show Ann that the written comments apply to the bookcase back.



**Figure 5.5** The Furniture Design Critic points out a design problem with evaluative feedback, and annotates Ann's design. A red arrow indicates lateral loads on the bookcase.

Ann understands this critique and revises her bookcase design, making the back part bigger to enhance stability. (Ann also adds several more shelves.) Looking again, the Furniture Design Critic sees that the problem has been resolved and finding no further serious issues, the Critic merely suggests more exploration (Figure 5.6): *“You are designing a bookcase. Explore other bookcase designs.”* But Ann is pleased not to have any outstanding problems, so she decides to stick with this design.

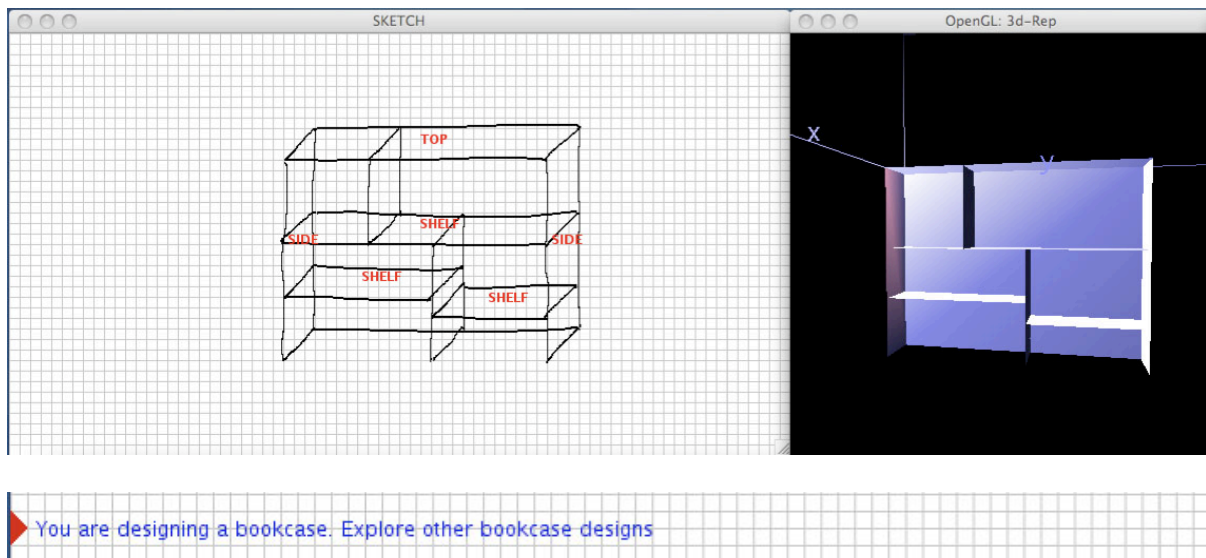


Figure 5.6 Finding no further issue to point out, the Critic merely suggests more exploration.

### 5.5.2. Vignette 2: Ben, a knowledgeable designer

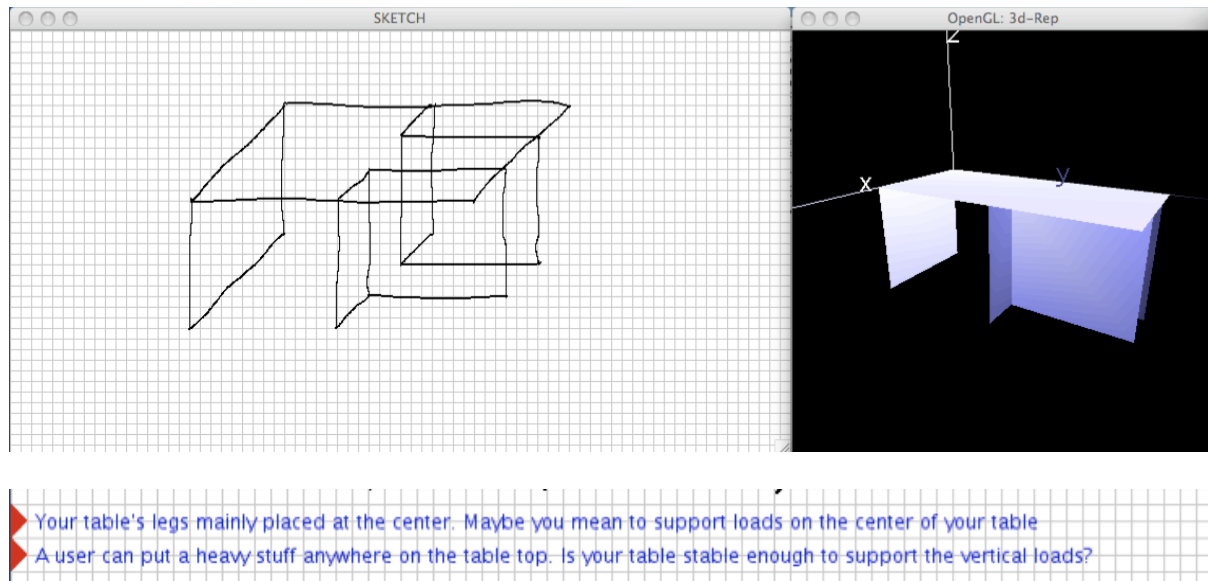
In this second vignette, in addition to considering the designer’s profile, the Furniture Design Critic also takes into account what critiquing methods have been previously used, then selects appropriate critiquing methods. Unlike with Ann, in this session with Ben I have added a set of critiquing rules that enables the Critic to select a different method, on the basis of what the critiquing methods it has

previously used. Feedback presented in different forms provides designers with opportunities to think differently about their designs. For example, evaluative feedback leads designers to think about how to repair a problem, whereas feedback presented using ‘introduction’ leads them to think about how to apply a new (or perhaps forgotten) idea into their designs.

In this session, Ben, a fairly knowledgeable designer, is working on a coffee table that several people can gather around. The Furniture Design Critic looks at Ben’s five-legged table design and notices several issues. Most critically, although the table has enough legs (five) to be stable, the placement of the legs suggests that the table might be unstable. The Critic offers feedback using the delivery types ‘interpretation’ + ‘introduction/ reminder’ and the communication modality ‘written comments’. Because Ben is quite knowledgeable, rather than presenting an evaluative critique (as it did with Ann), instead the Furniture Design Critic interprets his design. It says (Figure 5.7): *“Your table’s legs are mainly placed at the center. Maybe you mean to support loads only at the center of your table.”* The Critic also introduces an idea that Ben seems not to have thought about: *“A user can put a heavy load anywhere on the table top. Is your table stable enough to support a heavy object placed at the corner?”*

Feedback presented as ‘interpretation’ leads Ben to see his design from the Critic’s point of view. The Critic’s reading of Ben’s design may or may not be same as what Ben had in mind. The interpretation may help Ben see his design differently—specifically, from the Critic’s viewpoint. The form of ‘introduction/ reminder’ the critique provides a new idea that Ben may not have thought about: Ben will think about how to apply the idea to his design. In this case Ben will consider what

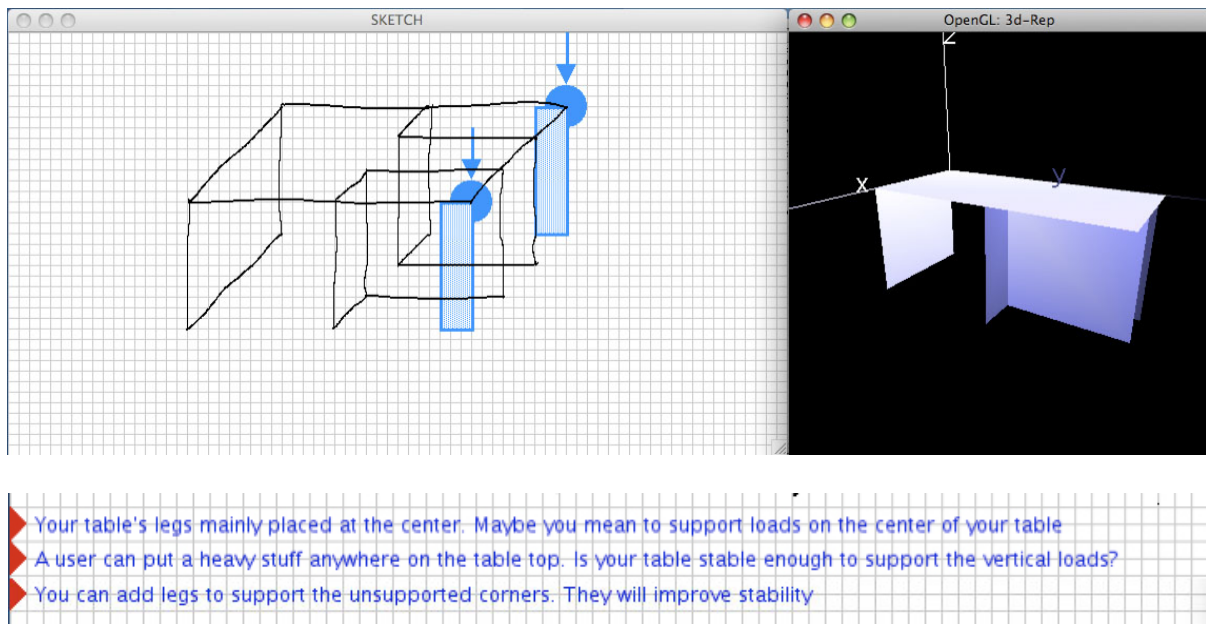
happens when a user puts a heavy load in different places on the table's surface, and more generally, the table's stability relative to vertical loads.



**Figure 5.7** The Furniture Design Critic observes and interprets Ben's table design. The Critic also introduces an idea that Ben has perhaps not considered.

Despite this feedback, Ben is confused, so he asks for more help. The Furniture Design Critic understands that its previous critique was ineffective, so it offers feedback using a slightly different critiquing method. It adds 'demonstration' to the delivery type, and 'graphical annotation' to the communication modalities (Figure 5.8). The Critic repeats its written comment (Figure 5.7), describing what Ben has done by interpreting his diagram and introduces an idea that Ben should think about in order to improve his design. But now, in addition to this previous feedback the Critic also demonstrates how to improve Ben's design. The Critic also marks on Ben's sketch to help him understand those two written comments: it draws two arrows and circles to indicate vertical loads

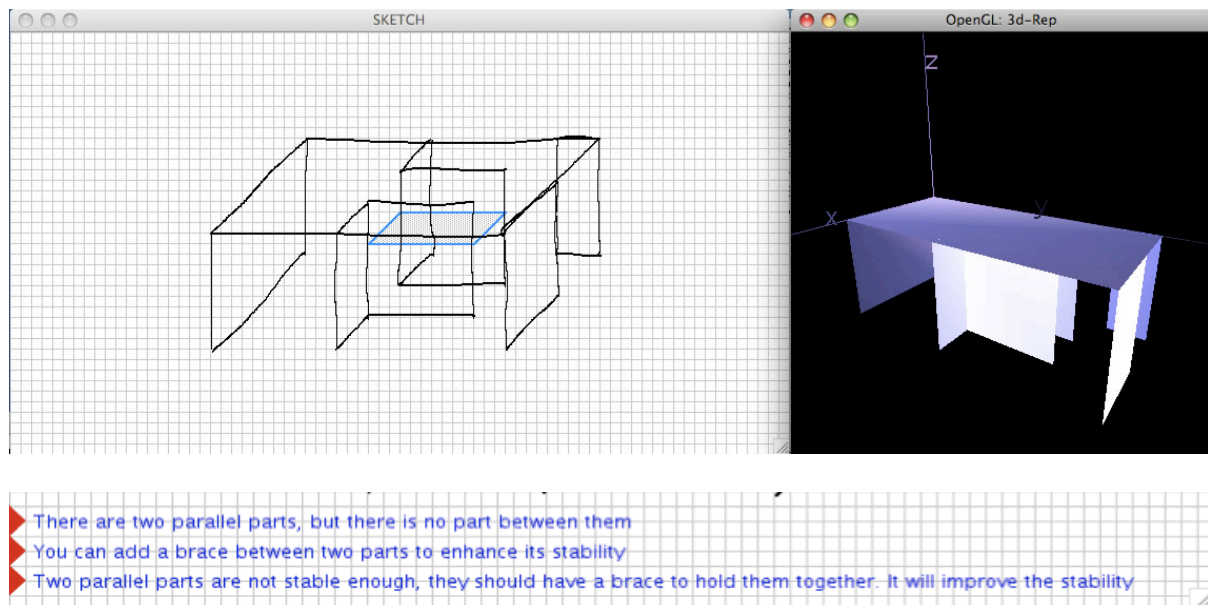
placed on the unsupported corners. The Critic also draws two leg parts in blue to show—graphically—how to resolve the problem of unsupported corners. And, in addition to marking up Ben’s sketch, the Critic offers the written comment *“You can add legs to support the unsupported corners. They will improve stability.”* This demonstration shows a solution to Ben’s stability problem.



**Figure 5.8** The Furniture Design Critic examines Ben’s table design and describes what it sees. It introduces an idea that Ben may not have thought about and demonstrates how to resolve a problem – two unsupported corners. It also annotates Ben’s diagram to help him understand the written comments.

Responding to the Furniture Design Critic’s suggestion, Ben adds legs at the unsupported corners. The Critic analyzes the design after Ben’s revision and sees that the problem has been resolved. However, it identifies another issue: Ben’s table has no brace between the legs, so it could be wobbly. Now, knowing what critiquing methods have worked with Ben, the Furniture Design

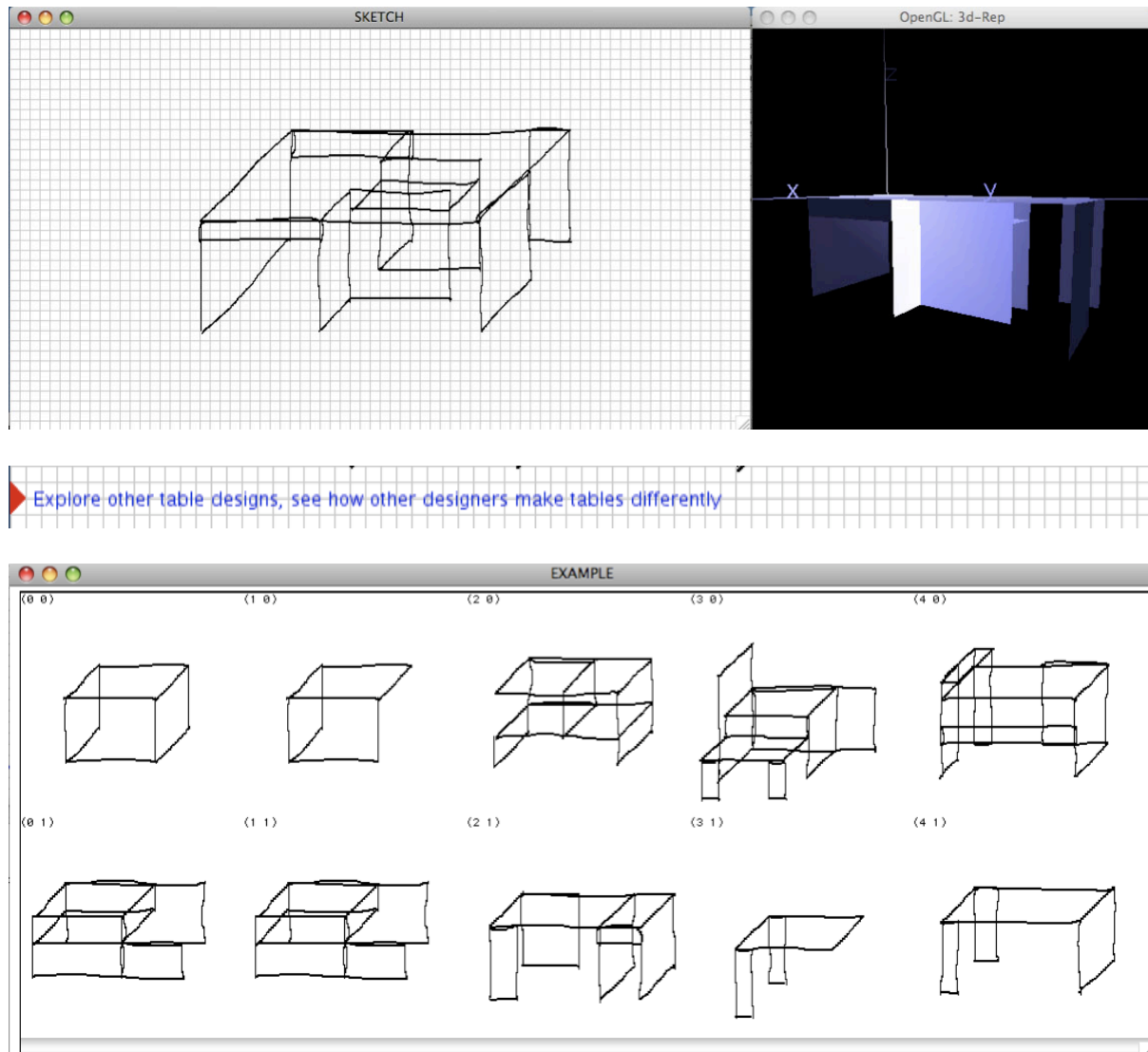
Critic again selects critiquing methods—the delivery types ‘interpretation’ + ‘demonstration’ + ‘evaluation’ and the communication modalities ‘written comments’ + ‘graphical annotations’ (Figure 5.9). It interprets Ben’s drawing, saying *“There are two parallel parts, but there is no part to hold them together.”* The program demonstrates, *“You can add a brace between two parts to enhance its stability.”* And it also writes the evaluative comment: *“Two parallel parts are not stable enough, they should have a brace to hold them together. The brace will improve the stability.”* The Critic also draws a brace in blue on Ben’s diagram to graphically demonstrate how to enhance the table’s stability.



**Figure 5.9** The Furniture Design Critic thinks aloud about Ben’s table design. It also demonstrates a brace by marking up Ben’s diagram and writes an evaluative comment.

Based on the Critic’s feedback Ben adds a bracing part between two parallel legs and two braces between other legs to enhance their stability. Now the Furniture Design Critic finds no

problems. It suggests more design exploration, *“Explore other table designs, see how other designers make tables differently.”* It shows some other table design examples (Figure 5.10).



**Figure 5.10** The Furniture Design Critic suggests more design exploration by presenting tables that other designers have made.

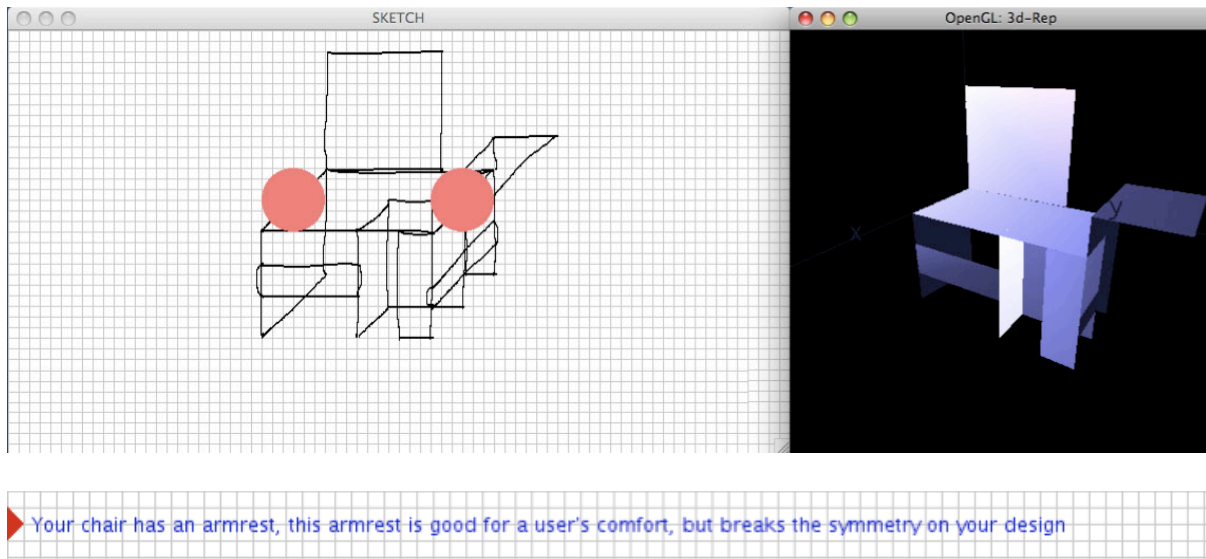
The examples that the Furniture Design Critic presents can help Ben develop alternative table designs, or to apply some ideas from the examples into his table design. So Ben goes to design another table.

### 5.5.3. Vignette 3: Claire

In this third vignette, the Furniture Design Critic the Critic considers the designer's profile, what critiquing methods were previously used, the interaction history with the designer, and the combination of delivery types and communication modalities.

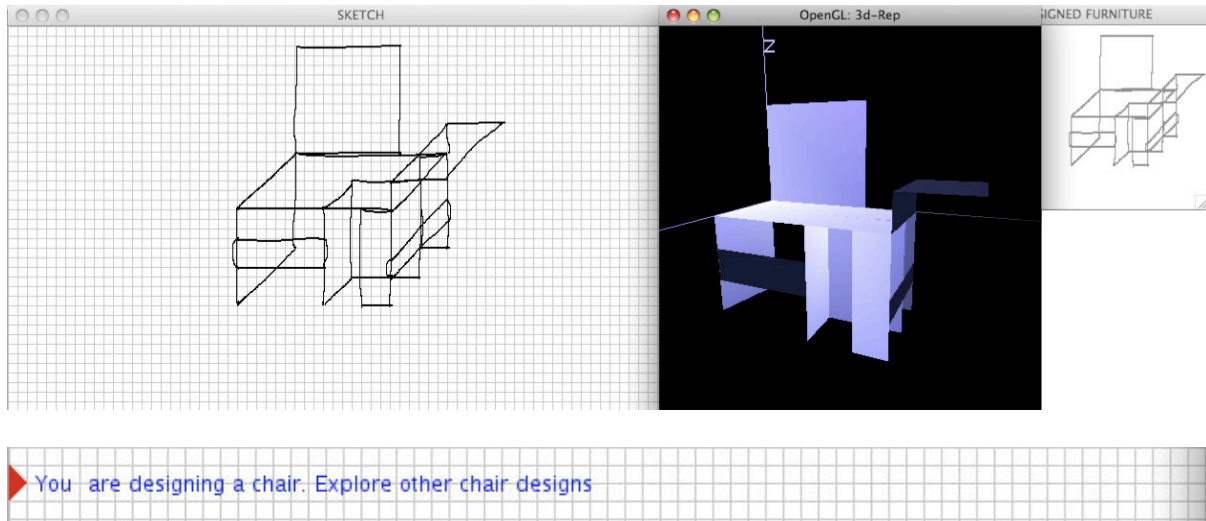
In this session the designer is Claire, a designer who has some experience, but who has a weak understanding of structural stability. The Furniture Design Critic sets a design task, a dining set composed of a table and two chairs. Claire can design any form of chairs and table she likes but they must stand by themselves, be strong enough structurally to support two people, and be comfortable to use.

Claire first decides to design the chairs. The Critic looks at Claire's design, notices that her chair has only one armrest, and selects appropriate methods. It first offers the encouraging positive evaluative comment: *"Armrests are a good idea because they make the chair more comfortable."* It also presents a negative opinion: *"but having only one armrest breaks symmetry on your design."* The Critic annotates Claire's diagram drawing two circles in red to indicate the symmetrical positions of armrests on her chair design (Figure 5.11).



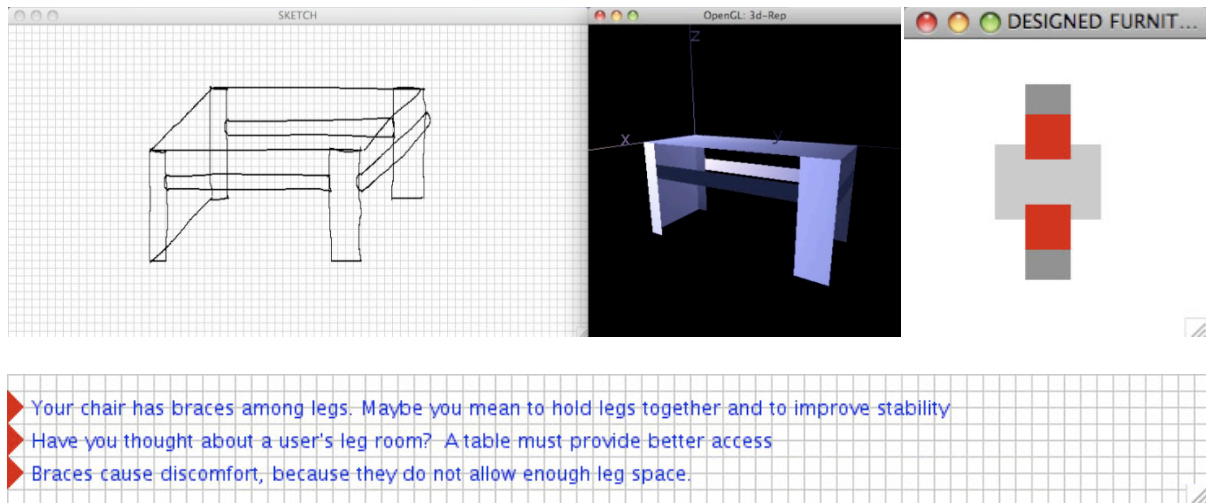
**Figure 5.11** The Furniture Design Critic points out that her chair has only one armrest. It makes graphical annotation on the positions of armrests.

Claire understands the feedback but she ignores it, because she wants to design a chair with only one, left, armrest—she tends to use only the left armrest. The Furniture Design Critic notices that Claire does not revise her design. However, it will not raise this issue again, because it knows it is relatively unimportant, not critical to her chair’s stability. The Critic finds no further problems in Claire’s design, so it suggests exploring alternative chair designs (Figure 5.12): *“You are designing a chair. Explore other chair designs.”* Claire, however, likes this design and keeps this chair for the dining set.



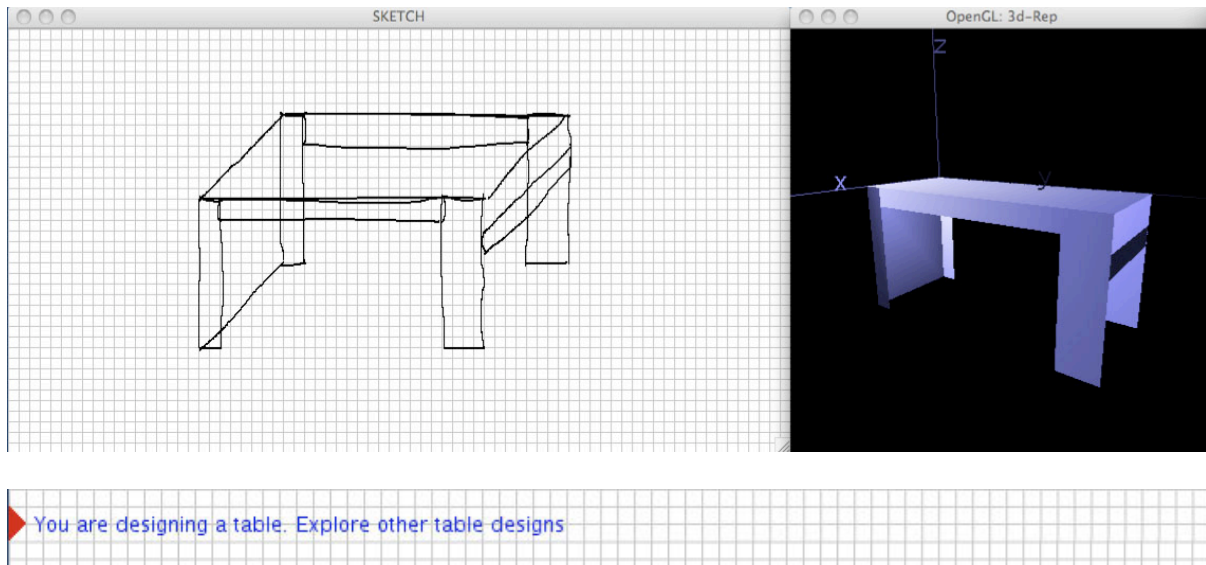
**Figure 5.12** The Furniture Design Critic suggests more design exploration. Claire decides to keep this chair design.

Now Claire begins to design the table. The Furniture Design Critic points out that the table must allow enough leg space for a user. It employs three delivery types – ‘interpretation’ + ‘introduction/ reminder’ + ‘evaluation’ and two communication modalities – ‘written comments’ + ‘image’ (Figure 5.13). It interprets Claire’s drawing, saying *“Your chair has braces among legs. Maybe you mean to hold legs together and to improve stability.”* The Critic introduces an idea, a concept of leg room, raising a question, *“Have you thought about a user’s leg room? A table must provide better access.”* It also writes the evaluative comment, *“Braces cause discomfort, because they do not allow enough leg space.”* It also offers a diagram that shows the plan view of her dining set composed of a table and two chairs (Figure 5.13, at right). In this plan view diagram, the center (light gray) rectangle represents a table and two darker gray rectangles represent chairs. The two red rectangles represent legroom for the two chairs. This diagram shows Claire why braces across the base of the table can cause discomfort.



**Figure 5.13** The Furniture Design Critic interprets Claire’s chair design, introduces a concept of leg room, and points out chair’s braces cause discomfort. It also presents an image to explain why braces are problematic.

Claire looks at the feedback and then moves the braces up, just below the tabletop, because she thinks that her table is probably unstable without these braces. She revises the table by moving the braces to make enough leg space. The Furniture Design Critic then finds no further problems, so it suggests more exploration (Figure 5.14): *“You are designing a table. Explore other table designs.”* But Claire decides not to change her current table design and stops her exploration there.



**Figure 5.14** The Critic suggests more design exploration

#### **5.5.4. Summary**

In the three sessions described in this chapter, the Furniture Design Critic program, equipped with different selection mechanisms, first identifies critiquing opportunities in a student's design, then decides which critiquing methods ("delivery types and communication modalities") to use in different critiquing conditions, and finally presents the feedback using the methods it chose.

In these sessions the Furniture Design Critic program selects critiquing methods by considering (1) the designer's profile, (2) what critiquing methods have been previously used with the designer, (3) the interaction history with the designer and (4) the combinations of delivery types and communication modalities. In the first session (Ann) it only looked at the designer's profile; in the second (Ben) it looked also at the methods that it had previously used to communicate with the

designer; and in the third session (Claire) the program looked at all four of the conditions listed above.

These critiquing sessions focused on describing the critiques presented using particular sets of critiquing methods. However, they did not reveal how—the mechanisms by which—the Critic (the Furniture Design Critic program) decides what critiquing method is appropriate in a certain critiquing situation. The following chapter describes the details of the framework of design critiquing and the mechanics of the Furniture Design Critic program that implements this framework.

## **6.A Computational Model of Design Critiquing: How the Furniture Design Critic works**

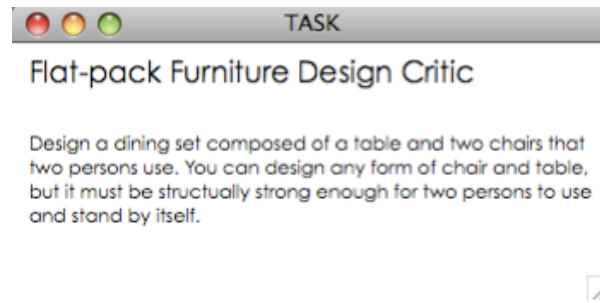
### **6.1. A Closer Look at the Vignettes**

The previous section (5.5 Furniture Design Critic: Three Vignettes) presented three vignettes that represent different critiquing sessions. The vignettes showed what the Furniture Design Critic program does to support designing. In each critiquing session the program used different mechanisms to select critiquing methods in reference to the critiquing conditions such as a designer's profile, critiquing methods previously used and the history of interaction with the designer. Before describing these mechanisms in detail in the following chapter (7 Implementation Details), here I briefly outline how a designer starts to work with the program, how the program makes inferences about the designer, and how the program detects critiquing opportunities, and

selects a method to deliver a critique. This chapter revisits the three vignettes in Section 5.5, explaining how the Furniture Design Critic does what it does.

The designer first logs in to the Furniture Design Critic. Because the designer has worked with the program before, the Critic retrieves its record of her previous design sessions from its database of design histories. While the designer was designing her furniture the program stored its interactions with her, in particular the delivery types and communication modalities it used to advise her about particular constraint violations. Using these stored interactions such as constraints that are violated and satisfied and critiquing methods that were previously used the program also made inferences about the designer. Therefore the history the program retrieves includes the designer's overall knowledge level, the method or methods that have proven effective, and specific weaknesses in the designer's knowledge. When she logs in, the program loads her recorded history, in order to select critiquing methods that are appropriate for her.

A designer can begin to work with the program in one of two ways. In the simpler way, the program provides a specific design task. For example, Figure 6.1 shows the design task that the program offers in the third critiquing session: "Design a dining set composed of a table and two chairs, ... etc."



**Figure 6.1** The ‘dining set’ task that the program has assigned in the third critiquing session.

However, the designer can also begin to design without the program first assigning a task. In this case, the program must try to recognize what the designer is doing by matching the design sketches with previously stored furniture designs. Each design is stored as a structure among furniture parts. For example, the program knows that a table is composed of a top and legs and that the top is placed on the legs. Once the program identifies what a designer intends to design, it can select relevant constraints, i.e., constraints that specifically apply to the design of a table.

A designer starts by sketching a furniture design using a stylus and a digitizing tablet. The sketch interface records the drawing, identifies its Cartesian coordinate system, and generates a 3D model. Then the program tries to recognize the object of design (e.g., table, chair, and bookcase) by analyzing the spatial relationships of the furniture parts and by comparing these relationships against stored furniture templates.

The program also identifies critiquing opportunities by comparing the analyzed spatial relationships against design knowledge stored in the program in the form of constraints. Each constraint—a unit of “design principle” is defined in terms of spatial relationships.

The program recognizes a violation if the proposed design satisfies a constraint's relevance conditions, but violates satisfaction conditions. When the program detects more than one critiquing opportunity it selects the most important violated constraint. The level of importance of a constraint measures its influence on the stability of the whole furniture piece. If several violated constraints have same importance, the program chooses to address any previously violated constraint. The program then reasons about which critiquing methods are appropriate for this constraint, under the given critiquing conditions. It considers the violated constraint, and the interaction history of the designer.

The following sections describe three different selection mechanisms for critiquing methods, corresponding to the three vignettes in Section 5.5. Each section first summarizes the critiquing example presented in the last chapter, then outlines the process that the Furniture Design Critic uses to select a critiquing method, and finally illustrates this process by tracing through the example.

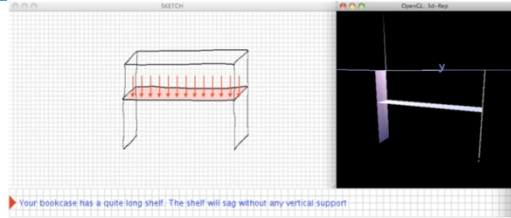
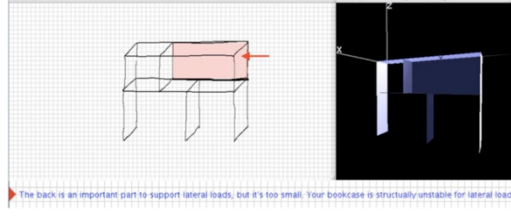
The Critic selects a particular set of critiquing methods by applying *Critiquing Rules* (See section 7.7). These *Critiquing Rules* decide the Critic's behavior, specifically the selection of critiquing methods. When these rules are changed, the Critic works differently. This section also describes what *Critiquing Rules* are used and applied in each critiquing scene. The reason that I describe *Critiquing Rules* is not that the *Critiquing Rules* mentioned here are prescriptively right and should be followed. Rather, I want to emphasize that *Critiquing Rules* can be easily modified and this change of *Critiquing Rules* leads me to change the way that the Critic selects a particular set of critiquing

methods. Therefore, the Furniture Design Critic program used to model and investigate different mechanisms of design critiquing model.

## 6.2. Vignette 1: Ann

Table 6.1 summarizes the two critiquing scenes in the first vignette about the beginning designer, Ann. In this example the Furniture Design Critic selects critiquing methods by looking at the designer's profile such as knowledge level, weaknesses, and the critiquing methods that have proven effective.

**Table 6.1 Summary of the first critiquing session, “Ann”**

	<p>• <b>Scene 1</b></p> <p>Delivery Types: <b>Evaluation</b></p> <p>Communication Modalities: <b>Written Comments + Graphic Annotations</b></p>
	<p>• <b>Scene 2</b></p> <p>Delivery Types: <b>Evaluation</b></p> <p>Communication Modalities: <b>Written Comments + Graphic Annotations</b></p>

If the program does not know the designer's profile (i.e., it has not previously worked with the designer), it simply chooses a predetermined sequence of methods. In this case the critic chooses delivery types in the following sequence:

1. interpretation,
2. introduction/ reminder,
3. example,
4. demonstration, and
5. evaluation.

The Furniture Design Critic chooses communication modalities in the following sequence:

1. written comments,
2. graphic annotation,
3. written comments + graphic annotation,
4. written comments + images,
5. written comments + graphic annotation + images.

Figure 6.2 outlines the program's reasoning process. The Furniture Design Critic is programmed to choose delivery types in this sequence to first offer critiques using facilitative delivery types (e.g., interpretation, introduction or example), because this feedback can prompt a designer to think about and improve the design. When the designer is unable to benefit from facilitative critiques, the critic changes to directive feedback (e.g., demonstration or evaluation).

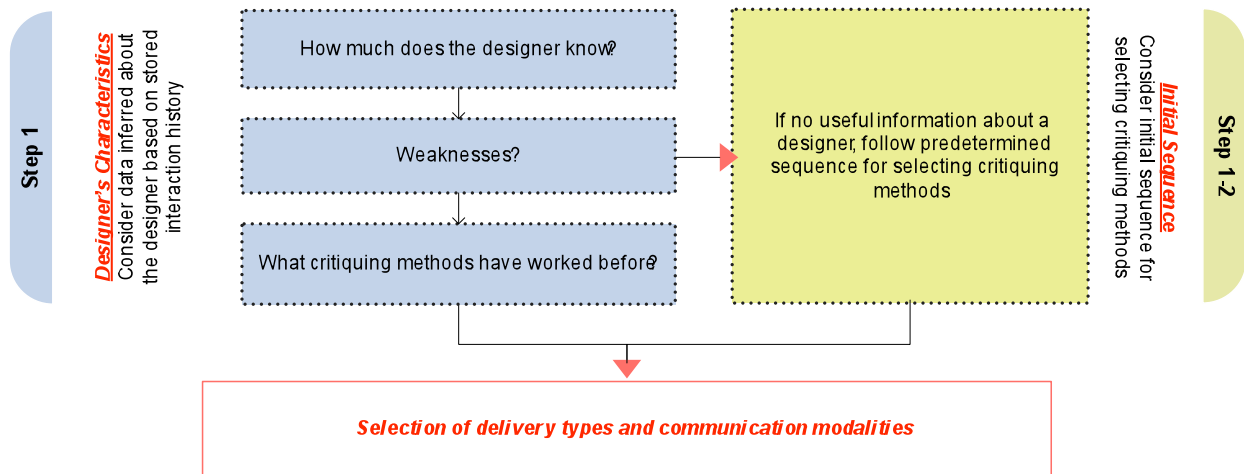


Figure 6.2 The Critic's reasoning process for selecting critiquing methods (Vignette 1)

Consider the two critiquing scenes in the first session with the beginning design student Ann (Table 6.1). In the first scene, the Critic analyzes Ann's profile—her overall knowledge level, specific strengths and weaknesses, and methods of critiquing that have proven effective (Step 1). Ann's knowledge level is low and her effective critiquing method is evaluation, so the program selects **evaluation** as a delivery type candidate. The Furniture Design Critic decides to offer feedback using **written comments and graphic annotations** because graphic annotations on Ann's diagram help her easily understand written comments. Figure 6.3 illustrates this process: the box represents a step of the reasoning process and summarizes what critiquing methods are selected as candidates and why. The final selected set of critiquing methods is **[evaluation]** and **[written comments + graphic annotations]**.

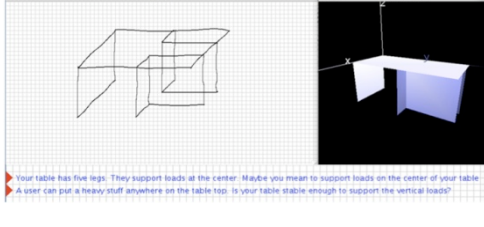
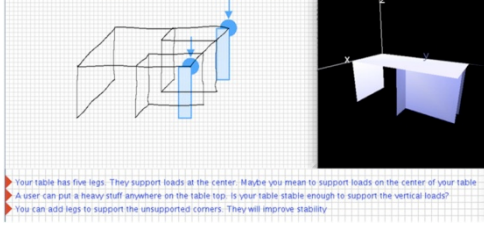
The second scene is no different: the Critic selects the same critiquing methods as in the first because Ann's profile has not changed. The final selected set of critiquing methods **[evaluation]** and **[written comments + graphic annotations]**.



### 6.3. Vignette 2: Ben

Table 6.2 summarizes the two scenes of the second critiquing session with Ben.

**Table 6.2 Summary of the second critiquing session, “Ben”**

	<ul style="list-style-type: none"> <li>• <b>Scene 1</b></li> </ul> <p>Delivery Types: Interpretation + Introduction/ Reminder</p> <p>Communication Modalities: Written Comments</p>
	<ul style="list-style-type: none"> <li>• <b>Scene 2</b></li> </ul> <p>Delivery Types: Interpretation + Introduction/ Reminder + Demonstration</p> <p>Communication Modalities: Written Comments + Graphic Annotations</p>

In this example, I have added a second step to the Furniture Design Critic’s reasoning process, shown in Figure 6.4. In Step 1, the Critic looks at the designer’s profile. If the program knows about a designer, it follows Step 2. However, if it has no information about the designer, then it chooses the next default method in the predetermined sequence, as in the previous vignette with Ann.

In Step 2, the Furniture Design Critic considers what critiquing methods have been used previously to address the violated constraint. The critic removes those methods from the candidate set to avoid repeatedly offering the same form of feedback on the same issue. If all candidates have

already been tried for this issue, then the Critic adds a new method to the candidates, choosing from among other methods not in the candidate set.

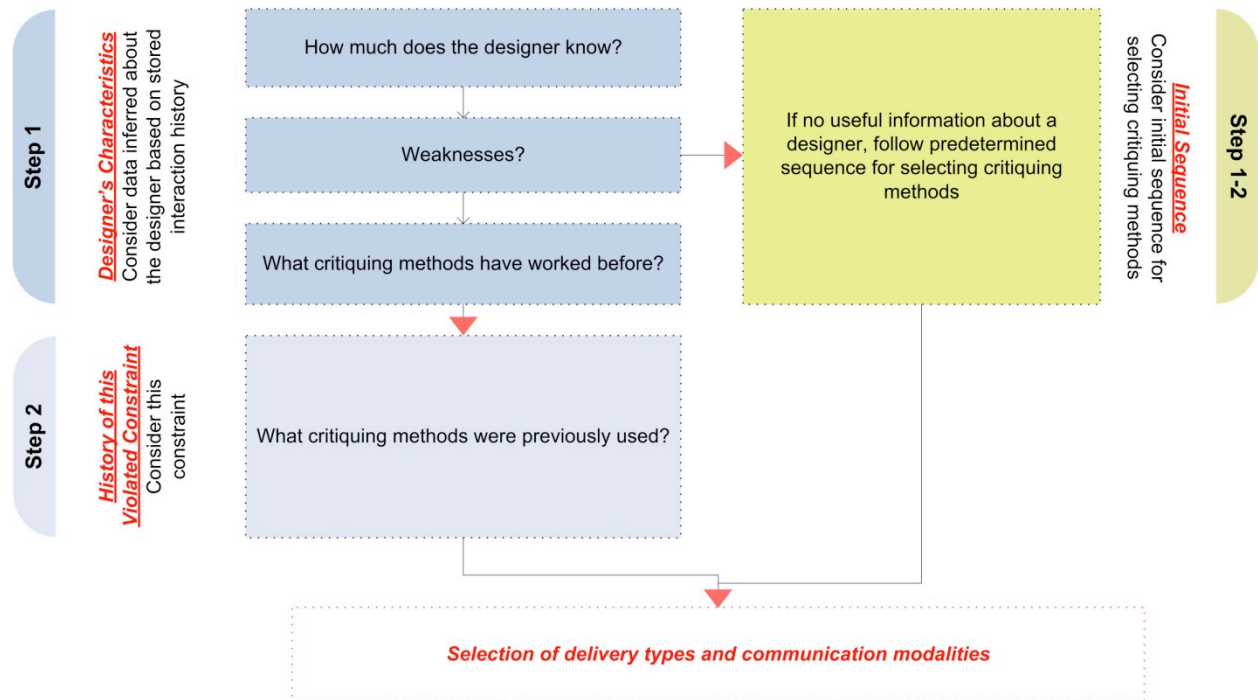


Figure 6.4 The Critic's reasoning process for the selection of critiquing methods (Vignette 2)

In the first scene of this vignette (Table 6.2), the Furniture Design Critic first considers Ben's profile (Step 1). Ben is a knowledgeable designer (his knowledge level is high), so the program selects **[interpretation + introduction/ reminder]** as delivery type candidates. The Critic selects **written comments** as a communication modality candidate. The critic always selects written comments as the fundamental modality. It then considers what critiquing methods have been used on the previous violations of the constraint in question (Step 2). This constraint has not been violated before, so the Furniture Design Critic selects **[interpretation + introduction/ reminder]**

and **written comments**. This reasoning process in the first scene is summarized in Figure 6.5. Each box in Figure 6.5 and Figure 6.6 represents a step of the reasoning process and summarizes what critiquing methods are selected as candidates and why they are selected.

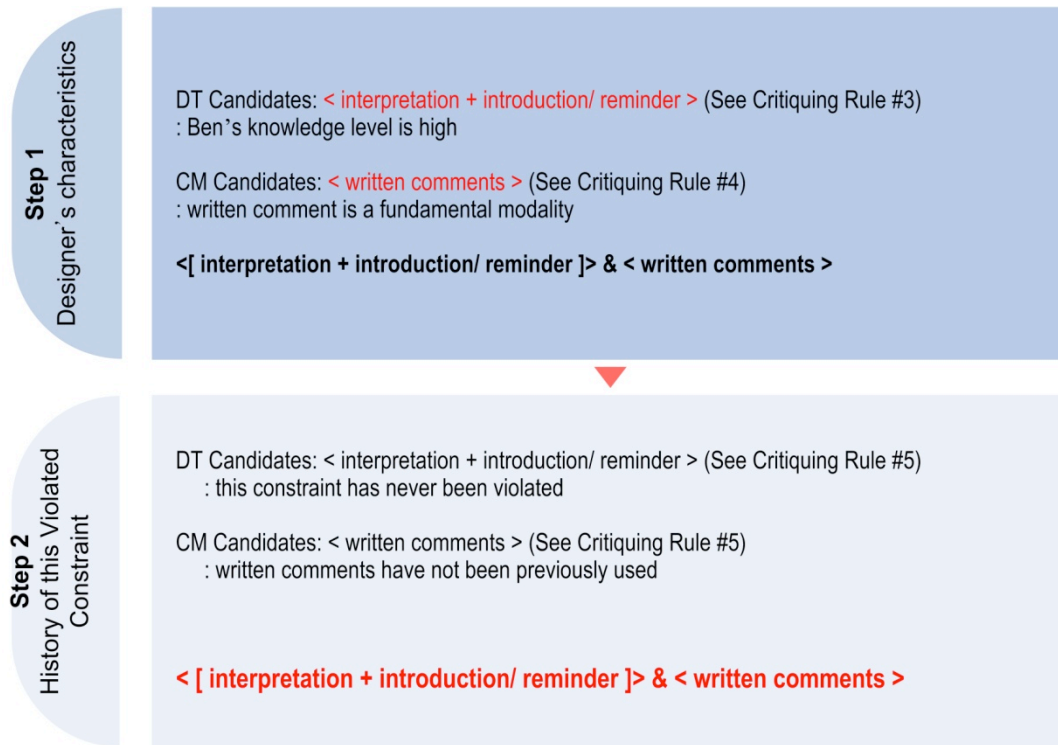


Figure 6.5 The Critic's reasoning process in critiquing Vignette 2, scene 1

In this Vignette 2, critiquing scene 1, the following critiquing rules are used. Figure 6.5 indicates when individual rules are used.

<b>Critiquing Rule #3</b>	<b>knowledgeable-designer</b>
[conditions]	if (and ( $R_{\text{violated}} \leq 0.2$ ) ( $R_{\text{violatedCritical}} \leq 0.1$ ))
[actions]	select delivery type candidate in step1 "interpretation" + "introduction"

#### *Critiquing Rule #4*

#### **fundamental-communication-modality**

[conditions]  
[actions]

always  
select communication modality candidate in step1 "written comments"

#### *Critiquing Rule #5*

#### **Never-been-violated-constraint-previously**

[conditions]

if (Number-of-violation-this-constraint = 0)

[actions]

confirm the selected delivery type candidate in step1  
confirm the selected communication modality candidate in step1

In the second scene (Table 6.2), the Furniture Design Critic again examines Ben's profile (Step 1). Ben's knowledge level is (still) high, so the program selects **[interpretation + introduction/ reminder]** as delivery type candidates and it selects **written comments** as a communication modality candidate. Next it considers the critiquing methods used on previous violations of this constraint. The delivery type candidate, **[interpretation + introduction/ reminder]**, was previously used with **written comments**. So the program adds **demonstration** to the set of delivery type candidates. This demonstration will show Ben how to resolve the constraint violation. The Critic also adds **graphic annotations** to the set of communication modality candidates. In the end, the selected methods are **[interpretation + introduction/ reminder + demonstration]** and **[written comments + graphic annotations]** (See Figure 6.6).

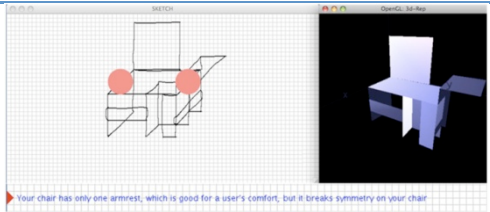
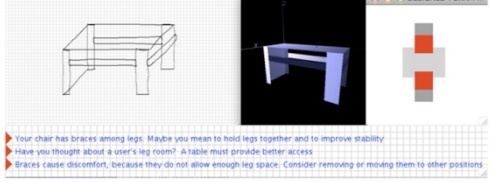


[actions]                      add “graphic annotations” into the selected communication modality candidate in step1

## 6.4.      Vignette 3: Claire

In the third and final vignette, the Furniture Design Critic critiques Claire. Table 6.3 summarizes this session.

**Table 6.3 Summary of the third critiquing session, “Claire”**

	<p>• <b>Scene 1</b></p> <p>delivery types: <b>evaluation</b></p> <p>communication modalities: <b>written comments + graphic annotations</b></p>
	<p>• <b>Scene 2</b></p> <p>delivery types: <b>interpretation + introduction/ reminder + evaluation</b></p> <p>communication modalities: <b>written comments + image</b></p>

I have added two additional steps to the program’s reasoning process: Now the Furniture Design Critic selects a critiquing method through four steps (Figure 6.7). In Step 1, the Critic looks at a designer’s profile—knowledge level, weaknesses, and effective critiquing methods. If the Furniture Design Critic does not know the designer’s profile, it follows the predetermined default sequence of methods described above.

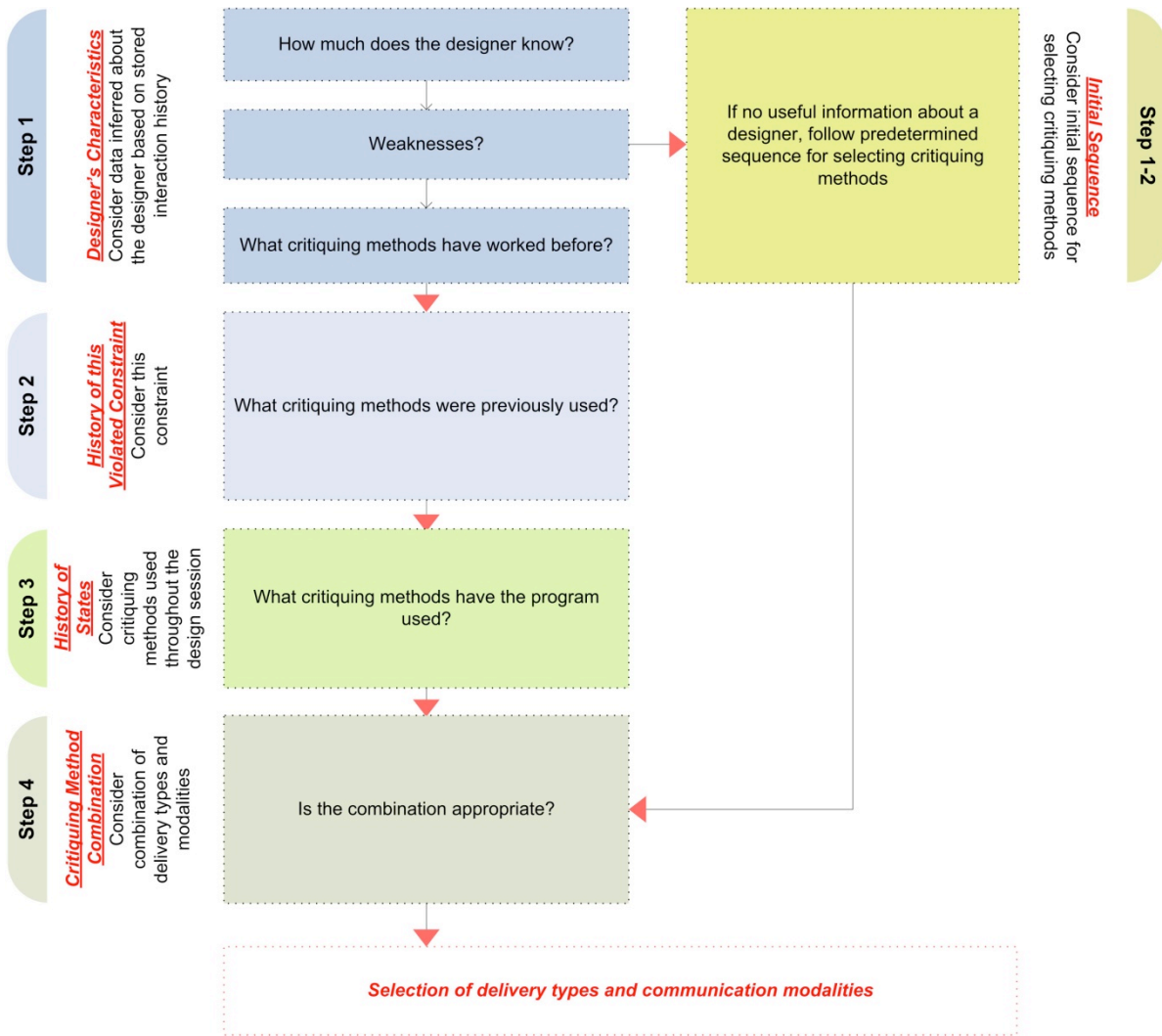


Figure 6.7 The reasoning process for the selection of critiquing methods (Vignette 3)

In Step 2, the Critic considers what critiquing methods have been used previously to address the violated constraint. It removes these methods from the candidate set to avoid repeatedly offering the same form of feedback on a constraint violation. If all candidates have already been

used for the constraint, then the Critic adds a new method to the candidates, choosing from among the methods that have not been used.

In Step 3 the Furniture Design Critic considers what critiquing methods have been used in all previous critiquing *states* regardless of the current constraint violation. A state represents a critiquing situation (See Section 7.10): (1) the constraint violated and selected for feedback; (2) the selected set of critiquing methods, (3) the selected delivery types; (4) the selected communication modalities; and (5) whether the selected critiquing methods were effective. For example, when a critiquing method has been used more than twice in previous states, the Critic selects a different method. When the Furniture Design Critic believes that the most effective critiquing method for the designer is evaluation; it will keep offering evaluative feedback. It is reasonable to communicate with the designer using only methods that have worked. However, this might also hinder the designer's opportunities to reflect on the design in different ways. In this step, therefore, the Critic 'experimentally' attempts to communicate with the designer in alternative ways by considering the previous *states*. In other words, this step performs an experiment to determine whether these alternative methods can help the designer improve the design, although the Furniture Design Critic has not selected these methods based on the critiquing situation, such as the characteristics of the designer. The result of this experiment is stored in the history of design states and then will influence the next selection.

Finally, in Step 4, the Furniture Design Critic considers the combination between delivery type and communication modality candidates to determine whether it is appropriate. For example, if the Critic has selected **demonstration** as a delivery type, it checks whether **graphic annotation or**

**image** is also selected as a communication modality. **Graphic annotation** and **image** can facilitate the designer's understanding of the demonstrated ideas. The Furniture Design Critic then uses the selected critiquing methods to present the critiques to the designer.

In the first scene (Table 6.3), the Furniture Design Critic first considers Claire's characteristics (Step 1). Each box in these Figure 6.8 and Figure 6.9 represents a step of the reasoning process and summarizes what critiquing methods are selected and why they are selected. Claire's knowledge level is quite low, so the Critic selects three delivery type candidates: **evaluation**, **demonstration**, and **[demonstration + evaluation]**. The Furniture Design Critic starts with **written comment** as a communication modality candidate, because it always uses **written comments** as a fundamental modality. The current candidates are **<evaluation, demonstration, [demonstration + evaluation]>** and **[written comments]**.

In Step 2, the Furniture Design Critic looks at what critiquing methods have been used on the previous violations of the constraint in question. This constraint was violated, but the selected candidates were not previously used. The Furniture Design Critic selects the first candidate, **evaluation**. It adds **graphic annotations** to the communication modality candidates, because written comments were previously used. Instead excluding **written comments**, the Furniture Design Critic adds **graphic annotations**, because **written comment** is the fundamental modality to communicate with a designer. At this point, the candidates are **[evaluation]** and **[written comments + graphic annotations]**.

In Step 3, the Furniture Design Critic analyzes what critiquing methods were previously used in the stored *states*. He has selected the same delivery type, **[evaluation]** twice. When the critiques offered in previous two *states* have both been unsuccessful, he may not repeat the same methods. In this case, the critiques using **[evaluation]** were all successful, so that the Critic decides to use **[evaluation]** again. At this step, the candidates are **[evaluation]** and **[written comments + graphic annotations]**.

Finally in Step 4, the Furniture Design Critic considers whether the combination among delivery type and communication modality candidates is appropriate. He thinks that the selected method candidates make appropriate combinations among delivery type and communication modality candidates. The final selected critiquing methods are **[evaluation]** and **[written comments + graphic annotations]**.

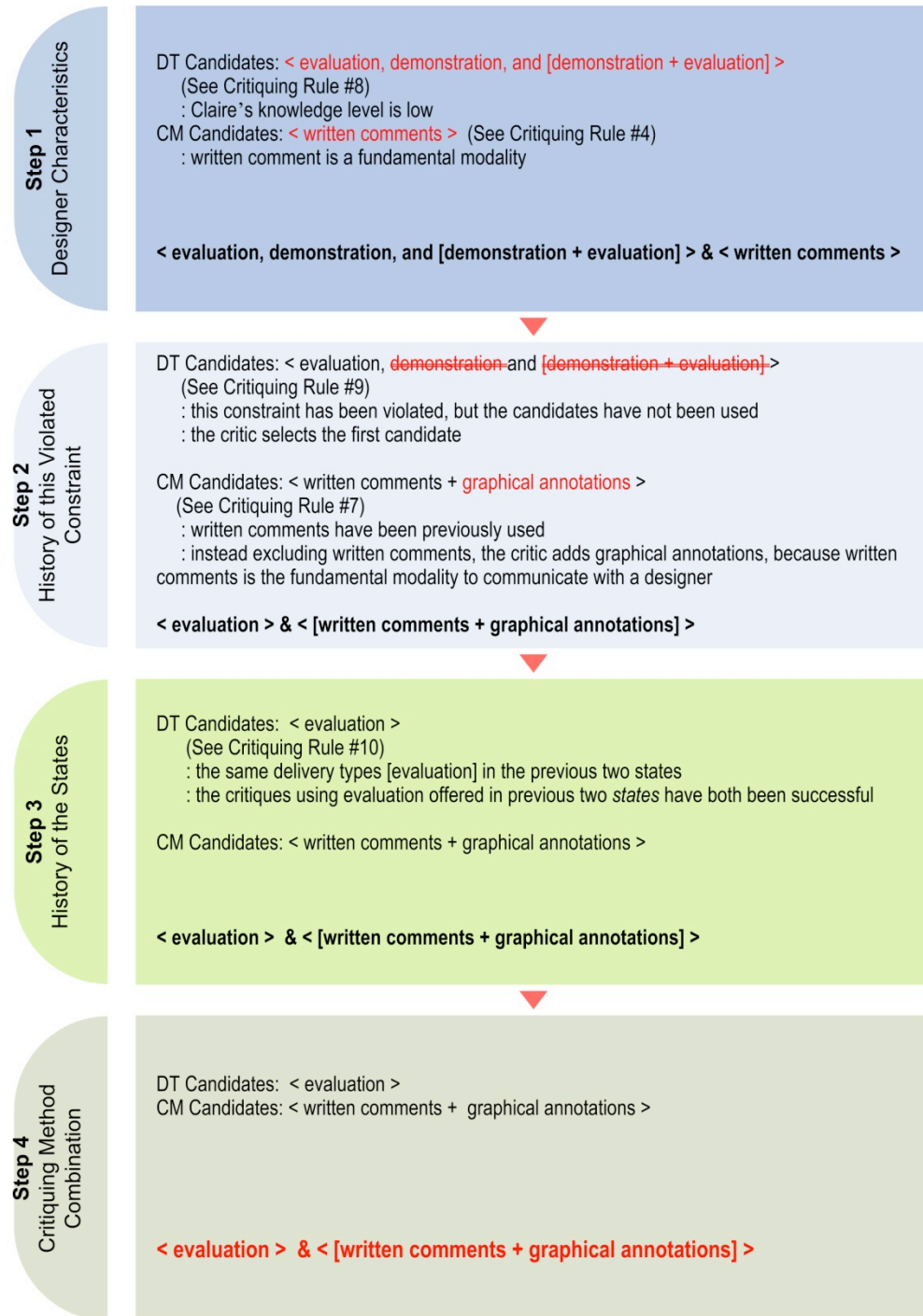


Figure 6.8 The reasoning process in critiquing scene 1 (Vignette 3)

In this Vignette 3, critiquing scene 1, the following critiquing rules are used.

*Critiquing Rule #8*      **notice-designer-third-mechanism**

```
[conditions]    if (and ( $R_{\text{violated}} \geq 0.4$ )
                    ( $R_{\text{violatedCritical}} \geq 0.5$ ))

[actions]      select delivery type candidate in step1,
                "evaluation", "demonstration", <"evaluation" + "demonstration">
```

*Critiquing Rule #9*      **used-delivery-type-candidates-not-contain-dt-candidate**

```
[conditions]    if (and (Number-of-violation-this-constraint > 0)
                        (Used-delivery-types do not contain all delivery-type-candidate-in-step1))

[actions]      select the first candidate from the selected delivery type candidate in step1
```

*Critiquing Rule #10*      **two-states-the-same-delivery-types-successful?**

```
[conditions]    if (and (the delivery type candidate in step2 = used-dt-previous-state)
                        (the delivery type candidate in step2 = used-dt-previous-state2))
                        (the used delivery type in previous state is successful)
                        (the used delivery type in previous state2 is successful))

[actions]       confirm and return the delivery type candidate in step2
```

Now consider the second critiquing scene (Figure 6.9). The Furniture Design Critic analyzes Claire’s characteristics in Step 1. Her knowledge level is low, so the Critic selects three delivery type candidates: **evaluation**, **demonstration**, and [**demonstration** + **evaluation**]. The Furniture Design Critic starts with **written comments** as a communication modality candidate. These are current candidates in this Step.

In Step 2, the Furniture Design Critic looks at what critiquing methods have been used on the previous violations of the constraint in question. The Critic selects the first candidate, evaluation, because the selected candidates were not used. He adds images into the communication modality candidates, because written comments and [written comments + graphic annotation] were used. At this point, the candidates are [evaluation] and [written comments + image].

In Step 3, the Furniture Design Critic analyzes what critiquing methods were previously used in the stored all *states*. The Critic has selected the same delivery types [evaluation] more than three times for Claire. In order to avoid repeatedly offering feedback using the same form, he adds other delivery types, [interpretation + introduction/ reminder]. The feedback presented using this combination, [interpretation + introduction/ reminder + evaluation] intends to help her understand why the raised issue should be resolved. At this step, the candidates are < [interpretation + introduction/ reminder + evaluation] > and < written comments + images >.

Finally in Step 4, the Furniture Design Critic considers whether the combination among delivery type and communication modality candidates is appropriate. It thinks that the selected method candidates make appropriate combination among delivery type and communication modality candidate. The final selected critiquing methods are < [interpretation + introduction/ reminder + evaluation] > and < written comments + images >.

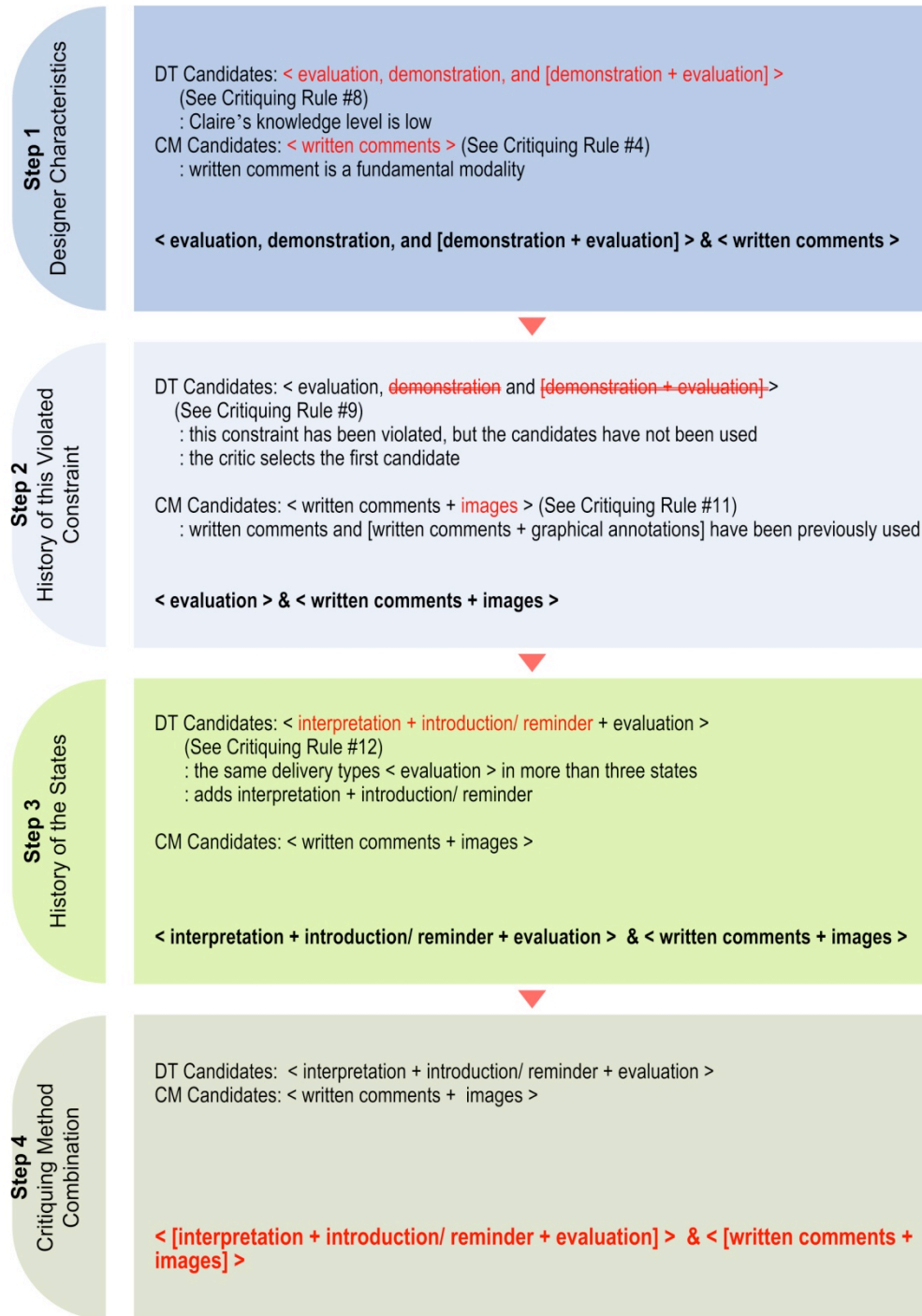


Figure 6.9 The reasoning process in critiquing scene 2 (Vignette 3)

In this critiquing scene of Vignette 3, the following rules are used. (I have omitted previously mentioned rules.)

*Critiquing Rule #11*      **previously-used-communication-modality-candidates-not-contain**

[conditions]	if (and (Number-of-violation-this-constraint > 0) (Used-communication-modality contain "written comments") (Used-communication-modality contain ("written comments" + "graphical annotation")))
[actions]	selects the communication modality candidate in step2, "written comments" + "images"

*Critiquing Rule #12*      **three-states-the-same-delivery-types?**

```
[conditions]    if (and (the delivery type candidate in step2 = used-dt-previous-state)
                        (the delivery type candidate in step2 = used-dt-previous-state2))
                        (the delivery type candidate in step2 = used-dt-previous-state3))
                        (the delivery type candidate in step2 = "evaluation"))

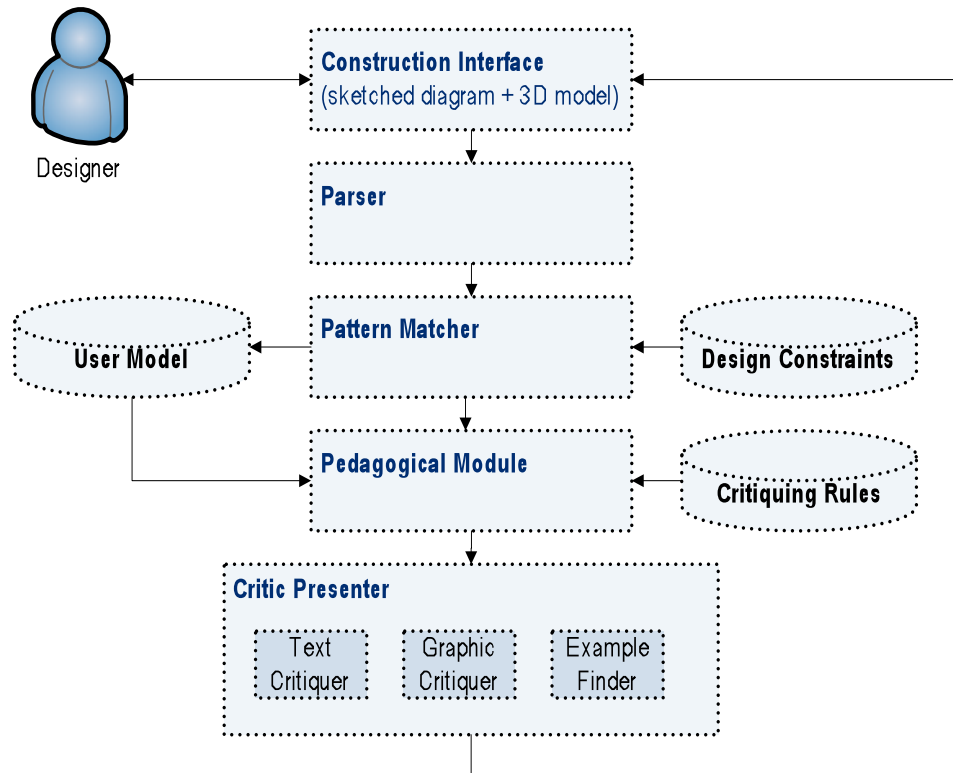
[actions]       add "interpretation" + "introduction" into the delivery type candidate in step2
```

This section has described three different selection mechanisms of critiquing methods for my development of the Furniture Design Critic program. Using these mechanisms I have analyzed the presented examples to explain how three different critics select particular sets of critiquing methods in consideration of the given critiquing situations. This is the main goal of this chapter. However, this has been described without explaining how a computer selects critiquing methods. The following chapter will describe what each system component does in detail and how the computer-based critic program works to execute these mechanisms in detail.

## 7. Implementation Details

The Furniture Design Critic adopts the typical system architecture of constraint-based intelligent tutoring systems. The Furniture Design Critic is written in MCL (Macintosh Common Lisp) using OpenGL to provide 3D models and the Lisa (Lisp-based Intelligent Software Agent) (Young 2007) production rule system to reason about a proposed furniture design using previously stored constraints.

The Furniture Design Critic is composed of several components: a Construction Interface, Parser, Pattern Matcher, Design Constraints, Critiquing Rules, User Model, Pedagogical Module, and Critic Presenter. Figure 7.1 shows these components and the information flow among them. This chapter follows the process shown in Figure 7.1 to describe what individual components do and how the Furniture Design Critic works.

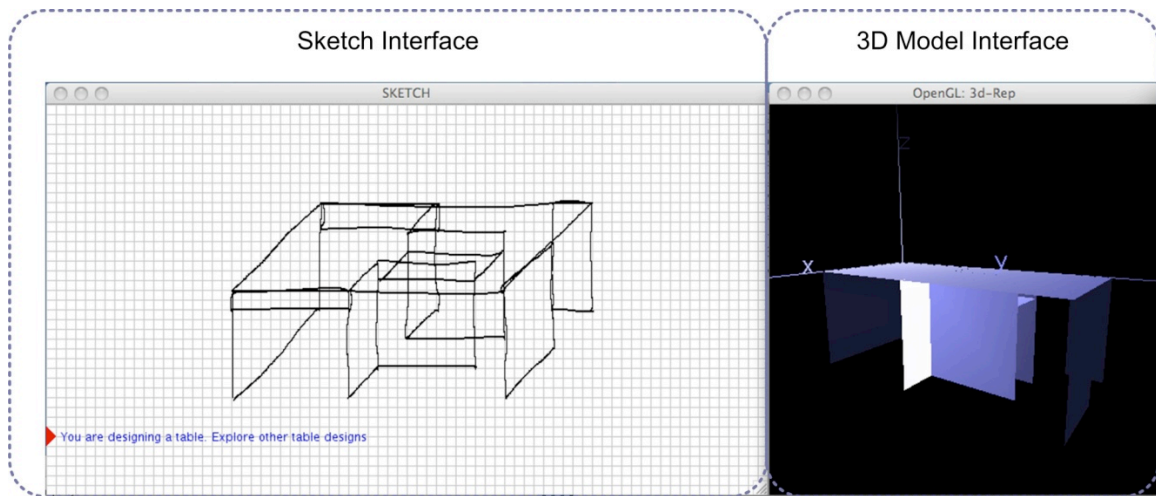


**Figure 7.1 Furniture Design Critic system components in the iterative construction-critiquing-repair cycle.**

## 7.1. Construction Interface

The *Construction Interface* is a design environment for flat-pack furniture. The designer (shown at left in Figure 7.1) sketches an axonometric diagram of a piece of furniture—for example a chair, bookcase, or table. The interface supports designing in several ways: constructing 3D models, decomposing them into parts with joints, and generating a cutout drawing ready for manufacture and assembly on a laser cutter computer-numerically-controlled tools. The interface consists of a Sketch window, a 3D Model window. Here also the system displays critiques. Figure 7.2 shows a

screen snapshot in which a designer is designing a table. The designer's sketch is shown at the left; the 3D model that the system has constructed from the sketch at the right.



**Figure 7.2 User Interface of Furniture Design Critic Program:** Sketch window (left); Model window (right).

## 7.2. Parser

Next in the information flow (in Figure 7.1) is the *Parser*. The *Parser* analyzes the designer's sketch to construct a 3D model with labeled parts. In the sketched diagram and the 3D model the *Parser* identifies the parts (e.g., width, height, which-plane, 2D coordinates, 3D coordinates, joints, etc.) and their spatial relationships or configuration (e.g., top-of, meeting (jointing), distance between two parts, parallel, between, etc.). The *Parser* creates a text file that stores a symbolic representation of the designed furniture. Later (in the *Pattern Matcher*) the Furniture Design Critic program uses this file to check whether the design violates domain principles stored previously in the system in the form of constraints.

For example, a designer draws a table with five legs (Figure 7.3). (What appears to be a single L-shaped leg at the corner nearest the viewer is actually recognized as two individual legs). The program stores the information about each furniture part including geometry (vertices and edges), dimensional data (length, 2D and 3D coordinates), a functional name (e.g., top, leg, side), and jointing (the connected faces). The Furniture Design Critic program generates the following symbolic representation for the Top of this table:

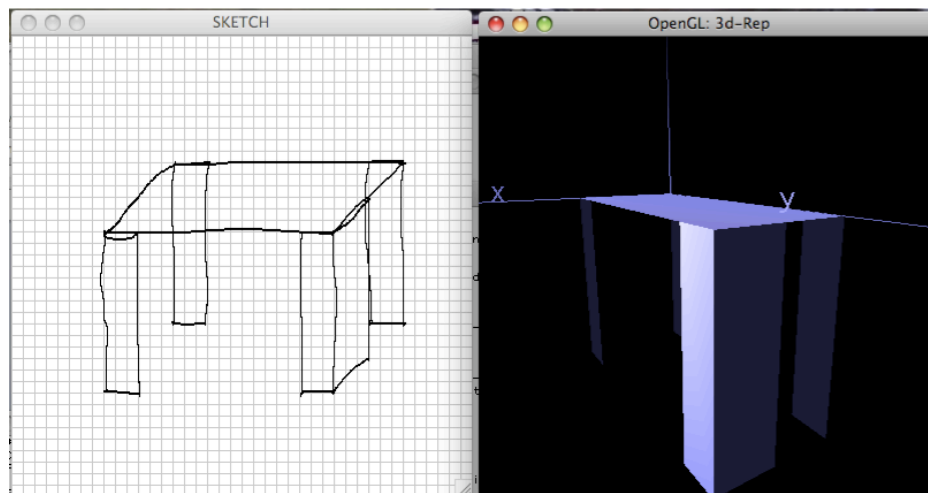
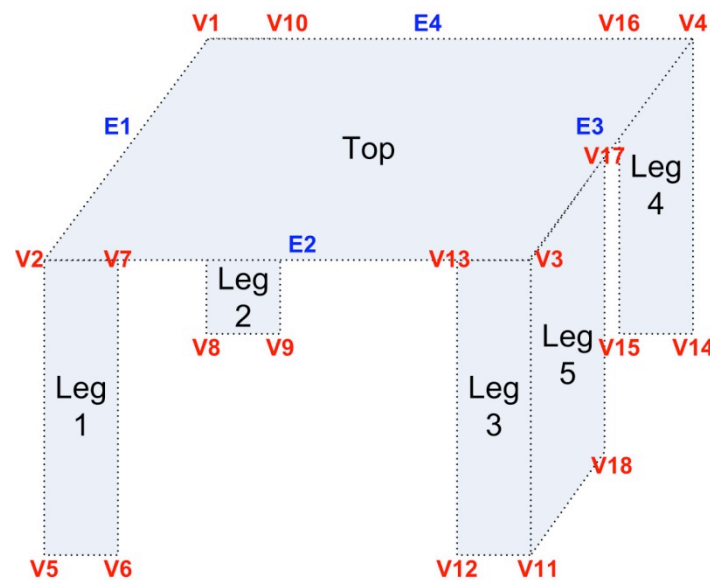


Figure 7.3 Sketch of a table with five legs

```
(furniturePart
(id "2.3.4.1.")
(vertices '(2 3 4 1))
(edges '(2 3 4 1))
(x-length 57.0) (y-length 190.0) (z-length 0) .....
(funcName "TOP")
(connected-faces '("5.6.7.2." "10.9.8.1." "11.12.13.3." "14.15.16.4." "11.18.17.3."))
(3dCoordinates '((57.0 0.0 0.0) (57.0 190.0 0.0) (0.0 190.0 0.0) (0.0 0.0 0.0))) ..... ))
```

The furniture part described here is the Top in the table shown in Figure 7.3, and diagrammed in Figure 7.4, which shows the labeling of the vertices and edges of each furniture part. The ID of each part is a unique string composed of its vertices' unique numerical identifiers —this part's identifier is **2.3.4.1**. In the above data structure, **vertices** '(2 3 4 1)' means that the top has four vertices: V2, V3, V4 and V1.

The descriptor **edges** '(2 3 4 1)' means that this furniture part is composed of four edges, E2, E3, E4, and E1. Also, '**connected-faces**' lists the other five furniture parts with which the top is connected "**5.6.7.2**" (leg1), "**10.9.8.1**" (leg2), "**11.12.13.3**" (leg3), "**14.15.16.4**" (leg4), and "**11.18.17.3**" (leg5). The variable **3DCoordinates** stores coordinate data of each vertex in 3D model world.



**Figure 7.4 A five-legged table: this diagram shows part names (in black) and geometric elements: vertices (in red) and edges (in blue)**

The program also records how these furniture parts are configured using a list of spatial relationships such as *parallel*, *between*, *top-of*, *jointing*, *distance*, etc. For example, two relationships of parts are stored as follows:

```
(top-of (faceA "2.3.4.1.") (faceB "5.6.7.2.)))    ;; faceA is placed the top of faceB
(parallel (faceC "5.6.7.2.") (faceD "10.9.8.1.)))  ;; faceC and faceD are in parallel
```

In the first, **top-of**, relationship **faceA** is the top part in the Figure 7.4. **faceB** is leg1. This Lisp form represents the relationship between two furniture parts: “*faceA is placed on top of faceB.*” In the second, **parallel**, relationship, **faceC** is leg1 and **faceD** is leg2 in Figure 7.4. This form says that “*faceC and faceD are in parallel.*” All these data of the design are written in a text file and are asserted into the *Pattern Matcher* to check the design against the previously stored design constraints.

### 7.3. Design Constraints

The system stores a set of *Design Constraints*, shown as a database in Figure 7.1. The constraints represent domain principles that designers need to know. The Furniture Design Critic program uses two types of design constraints: (1) structural constraints that specify the allowed structure of furniture parts and (2) functional constraints that specify allowed functions for certain parts of a piece of furniture. The structural constraints specify valid configurations of furniture parts and are used to identify structural problems in designers’ solutions. They vary from simple constraints such as “a long shelf must be supported from the middle”, to more complex constraints such as “a table’s brace must be placed to allow enough leg space.” Simple constraints only deal with a single furniture

piece; complex constraints identify problems in a furniture ensemble, such as a table and a chair. For example, the complex table-chair constraint checks whether a table has enough leg space by considering the position of a table's brace and the width and height of the chair that a designer has previously designed and stored. Functional constraints specify the functions of certain parts or a whole piece of furniture. For example, functional constraints include: "a chair may have an armrest" and "a horizontal bracing part parallel to the chair seat may be used as a shelf".

The program uses these constraints to check the proposed furniture designs. The following shows the data structure of each constraint and what data is stored in each slot.

```
(defclass constraint ()
  ((ID) ; numerical ID
    (name) ; brief description of a constraint
    (importance) ; define an importance of this constraint based the potential influence
    (relevance-conditions) ; relevance conditions
    (satisfaction-conditions) ; satisfaction conditions
    (critique-delivery-types) ; written comments using five delivery types
    (critique-modalities))) ; function calls to offer feedback using other two modalities, graphical
                           annotations and images
```

Each constraint consists of a unique ID, a name that gives a brief description of the constraint, a relevance condition, a satisfaction condition, its importance, written comments using the five different delivery types, function calls to present feedback using other two communication modalities. A constraint has a predefined importance level from 1 to 3 based on its potential influence on the stability of the whole furniture piece.

Each constraint data structure has two slots relevant to offering feedback in multiple methods: (1) *critique-delivery-types* and (2) *critique-modalities*. The *critique-delivery-types* item stores pre-defined written comments for the constraint in five different delivery types. The following shows the delivery type text strings for the constraint in a bookcase design that checks whether a back part is large enough to support lateral loads.

**(Interpretation** – “Your bookcase is composed of two sides, a shelf, a top and a back. The two sides and the back are vertical structural components for loads.”)

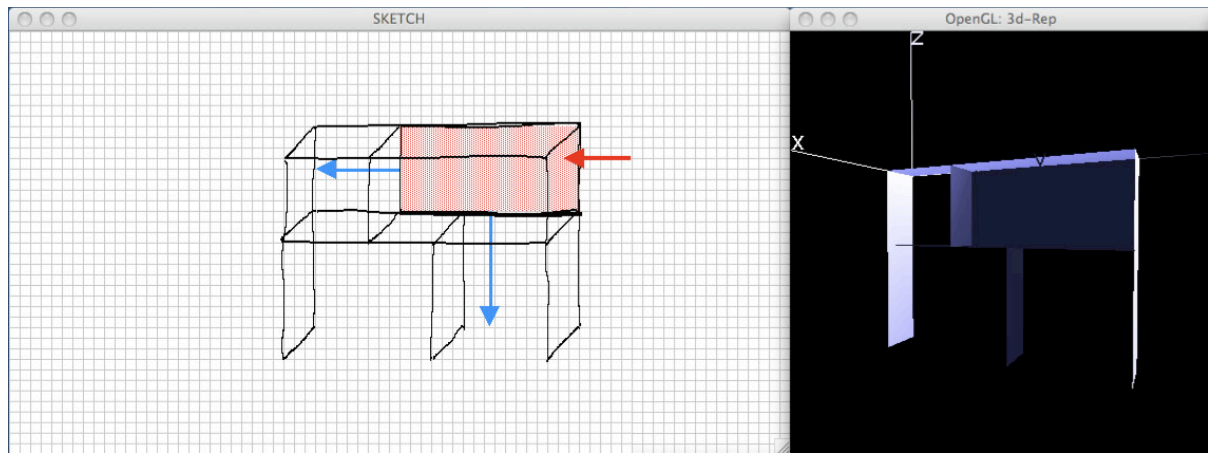
**(Introduction/Reminder** – “Is your back part big enough to support lateral loads?”)

**(Example** – “Look at other bookcases with backs.”)

**(Demonstration** – “Make the back part bigger as shown (drawing a bigger back part).”)

**(Evaluation** – “The back is an important part to support lateral loads, but it’s too small. Your bookcase is structurally unstable for lateral loads”))

The *critique-modalities* item stores a list of calls to routines that deliver feedback. The Furniture Design Critic delivers feedback using the selected communication modalities by executing these routines. For example, the Critic program calls a function to annotate a designer’s sketch by painting parts red that violate a constraint; displaying graphic icons such as arrows to indicate a load placed on a furniture part; and retrieving and presenting images of relevant examples. Figure 7.5 shows the result of the program calling three functions stored in the *critique-modalities* slot: highlighting a relevant back part in red, drawing a red arrow to indicate lateral loads, and drawing two blue arrows to indicate how the back part should be enlarged.



**Figure 7.5 A book case design: this design violates a constraint that checks whether the back is large enough to resist lateral loads. The system offers feedback using the demonstration delivery type.**

#### 7.4. Pattern Matcher

The *Pattern Matcher* (see Figure 7.1) compares the symbolic representation that the *Parser* generates against the *Design Constraints* in order to detect opportunities for critiquing. The *Pattern Matcher* takes the text file that the *Parser* has generated and checks whether the proposed design satisfies all relevant design constraints. The program determines that a constraint is violated if the proposed design satisfies the constraint's relevance condition but violates its satisfaction conditions.

For example, in Figure 7.6 the designer has drawn a bookcase with a long shelf with no support in the middle. This bookcase satisfies relevance conditions of a stored constraint: (1) the designer is designing a bookcase; and (2) the bookcase has a long shelf with width-to-length ratio greater than a threshold (e.g., 1:5). The program notices that the design satisfies the constraint's relevance conditions, so it checks whether it satisfies the satisfaction conditions. The satisfaction conditions are that an additional part supports the middle of the long shelf. As there is no middle

support, the program determines that the shelf is problematic and identifies why; in this case that ‘the long shelf is structurally unstable when books are placed on the shelf’.

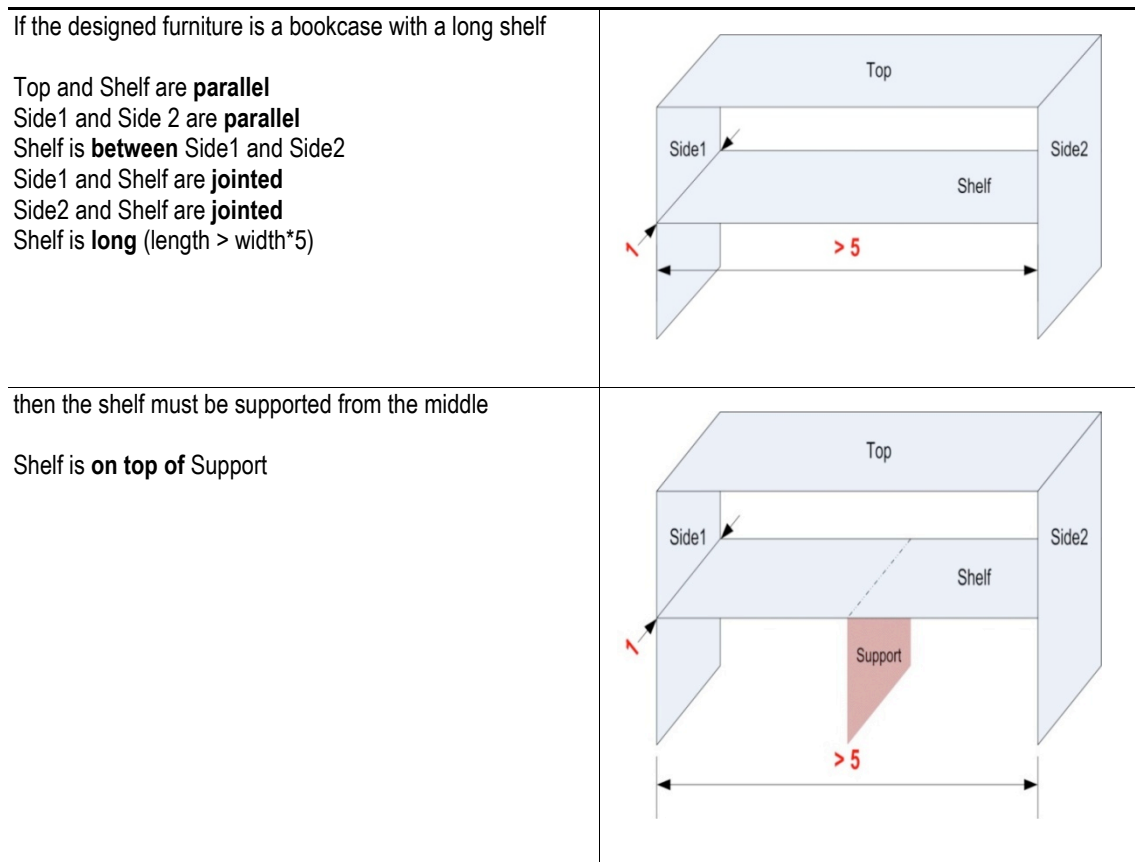


Figure 7.6 Pseudo-code and diagram showing a violated constraint.

## 7.5. User Model

The program has a two-part *User Model*: a short-term user model and a long-term user model, shown as a database in Figure 7.1. Whenever a designer asks for a critique, the program analyzes the designer’s work using the *Pattern Matcher* and then creates a short-term model. This short-term user

model stores the reasoning output of the *Pattern Matcher*, namely the set of violated and satisfied constraints. The data structure of each violated constraint is as follows:

```
(defclass ViolatedSatisfiedConstraint ()  
  ((constraint-ID) ; the unique constraint number  
    (violated-or-Satisfied) ; if this constraint is violated, V; if this constraint is  
    satisfied, S  
    (violatedNum) ; how many times this constraint is violated  
    (relevantParts) ; what furniture parts violate/satisfy this constraint  
    (usedCritiquingDeliveryTypes) ; used critiquing delivery types  
    (usedCritiquingModalities))) ; used critiquing communication modalities
```

When a constraint is violated or satisfied, the program stores this constraint by instantiating the class **ViolatedSatisfiedConstraint**. This **ViolatedSatisfiedConstraint** object records the following data:

- (1) the unique identifier indicating which constraint has been violated or satisfied;
- (2) whether this constraint is violated (V) or satisfied (S);
- (3) how many times it has been violated or satisfied;
- (4) which furniture parts are involved in violating or satisfying the constraint;
- (5) the delivery types previously used to offer feedback for this constraint; and
- (6) the communication modalities previously used to offer feedback for this constraint.

A violated constraint represents a design principle that a user has not yet mastered or an opportunity that a user has failed to notice. If the user violates the same constraint several times, the program infers that the designer has difficulty applying that design principle despite having been informed about it. The program also stores which parts of the design are relevant for each violated constraint. Then the *Critique Presenter* uses this list of parts to annotate the designer's diagram.

In the example above concerning the size of the bookcase back (Figure 7.5), the relevant parts are the back and the two sides. The program highlights the back and draws arrows between the

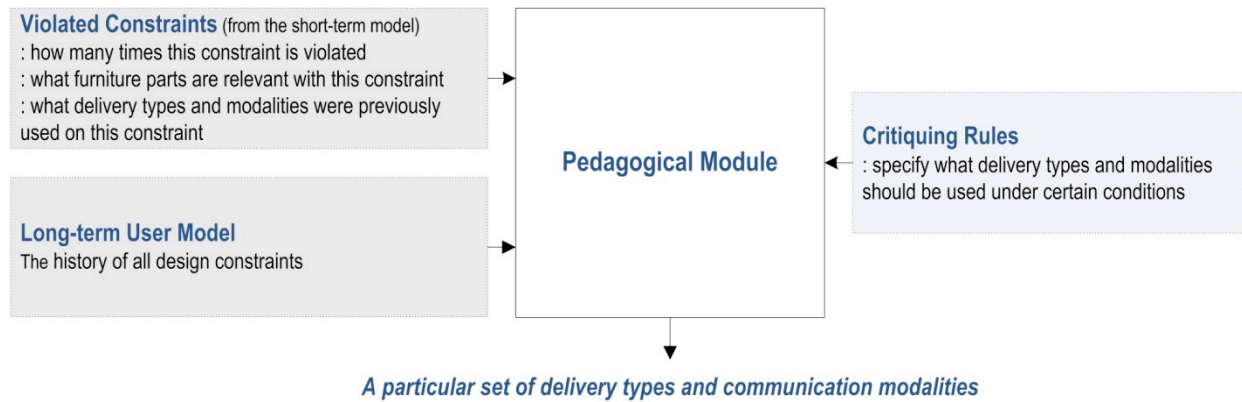
back and the sides by calling functions that are listed in the constraint's *critique-modalities* slot. The function computes appropriate positions for the two arrows and draws them on the designer's sketch to indicate how much the back part should be enlarged. The short-term user model also stores what delivery types and modalities have been previously used to offer critiques on a violated constraint. If the constraint is violated again, the program selects other delivery types and communication modalities by applying the *Critiquing Rules*.

The long-term user model stores the history of violated and satisfied constraints and history of all critiquing sessions, namely *States* (See Section 7.10). Using this history of all constraints including the violated and satisfied, the *Pedagogical Module* makes an inference about the designer (See Section 7.6). The *User Model* data structure is as follows:

```
(defclass UserModel ()
  ((designer-ID)           ; the unique ID of a certain designer (name)
   (history-ViolatedSatisfiedConstraints) ; history of all constraints including violated and satisfied
   (states)))             ; history of all critiquing sessions, states
```

## 7.6. Pedagogical Module

The *Pedagogical Module* is at the heart of the Furniture Design Critic. It is this program that selects the delivery type and communication modalities for a critique based on the inference results. As shown in Figure 7.1 the *Pedagogical Module* is executed after the *Pattern Matcher* has identified constraint violations in the design. The *Pedagogical Module* employs the *User Model* (short-term and long-term) and selects delivery types and communication modalities by applying *Critiquing Rules*.



**Figure 7.7 Pedagogical Module and the data involved for selecting critiquing methods**

The *Pedagogical Module* takes as input

- (1) data of a violated constraint from the short-term user model,
- (2) the history of all constraints in the *User Model*, and
- (3) the *Critiquing Rules*.

In other words, the *Pedagogical Module* considers the violated constraint and the *User Model* and chooses a particular critiquing method by applying *Critiquing Rules*. When a proposed design violates several constraints (i.e., the short-term user model contains more than one constraint), the program selects a constraint based on its importance and the number of times it has been violated. Between two equally important constraints, the program selects the one that has been violated most often. The *Pedagogical Module* then considers the chosen constraint from the short-term user model, makes an inference about the designer based on the long-term user model (the history of all *Design*

*Constraints*), and selects critiquing methods by applying the *Critiquing Rules*. I have briefly described what the *Pedagogical Module* does. Now let me explain how the *Pedagogical Module* executes the selection mechanisms of critiquing methods in following sections (Sections 7.6.1. – 7.6.3.). The *Pedagogical Module* analyzes the data of a violated constraint from the short-term user model and makes an inference about the designer using the data in the *User Model* (e.g., knowledge level, weaknesses, effective critiquing methods). These inference data correspond to critiquing conditions. The *Pedagogical Module* creates a text file that stores a representation of these critiquing conditions. It uses this file to make a decision about which critiquing methods are appropriate in each situation.

### 7.6.1. Selecting which feedback should be offered first

If a design violates more than one constraint the *Pedagogical Module* must decide in what order to deliver feedback. It addresses more important issues first. When several constraints have the same importance, the program chooses to address the previously violated issue. For example, suppose the *Pattern Matcher* has presented the program with the following short-term model (a list of violated and satisfied constraints):

```
*short-term model*                                ;; a list of ViolatedSatisfiedConstraints
(((constraint-ID 4)                                ;; the constraint, long-shelf-without-supporting-part (importance 1)
  (violated-or-satisfied "V")                      ;; a bookcase design violates this constraint #4
  (violatedNum 2)                                   ;; this user has violated this constraint two times
  (relevantParts '(shelf1))                        ;; the stored parts violate this constraint
  (usedCritiquingDeliveryTypes '(interpretation))  ; the used delivery type is 'interpretation'
  (usedCritiquingModalities '(written comments)))  ; the used communication modality is 'written comments'

((constraint-ID 12)                                ;; the constraint, lateral-loading-issue (importance 0)
  (violated-or-satisfied "V")                      ;; a bookcase violates this constraint #12
  (violatedNum 1)                                   ;; this constraint is violated first time
  (relevantParts '(top, side1, side2))             ;; the stored parts violated this constraint
  (usedCritiquingDeliveryTypes '())
```

```

(usedCritiquingModalities ' ( )))

((constraint-ID 23) ;; the constraint, recognition-bookcase-one-shelf (importance 3)
 (violated-or-satisfied "S") ; a bookcase satisfies this constraint #23
 (violatedNum 1) ;; this constraint is satisfied first time
 (relevantParts '(top, side1, side2, shelf1)) ;; the stored parts violated this constraint
 (usedCritiquingDeliveryTypes ' ( ))
 (usedCritiquingModalities ' ( )))

```

The *Pedagogical Module* must decide which among the constraints stored in the short-term user model to address first. It chooses the second constraint (ID #12), because it is the most important. Next, if the first and third constraints have the same importance level, the program chooses the first constraint (ID #4), because it has been violated before.

### 7.6.2. Making an inference about a designer based on the User Model

The program takes the history of all constraints including the violated and satisfied (the *User Model*) and infers (1) how much the designer knows about flat-pack furniture design; (2) which type of constraints that the designer has trouble with and what types of constraints the designer handles properly; and (3) which critiquing methods work well for that designer.

Let me describe how the program makes an inference about the designer. The program infers a designer's overall knowledge level based on the history of all design constraints that the program knows. It represents the designer's knowledge level by a pair of ratios ( $\mathbf{R}_{\text{Violated}}$ ,

$R_{\text{ViolatedCritical}}$ ). When both ratios are zero (0, 0), then the designer has violated no constraints and therefore has evidently mastered all the domain knowledge stored as constraints in the system<sup>9</sup>.

$R_{\text{Violated}}$  in Eq. (7.1) represents how much a designer does (or does not) understand about the design space represented by the set of constraints. The higher  $R_{\text{Violated}}$ , the less the designer knows.  $R_{\text{ViolatedCritical}}$  in Eq. (7.2) represents the designer's ignorance of important, or critical, constraints. The higher  $R_{\text{ViolatedCritical}}$ , the less a designer knows about the fundamental knowledge of the design domain. Critical constraints with high importance (level 1) are those that must be satisfied for designers to develop stable furniture.

$$(7.1) \quad R_{\text{Violated}} = (\text{number of constraints violated} / \text{number of all constraints the program knows})$$

$$(7.2) \quad R_{\text{ViolatedCritical}} = (\text{number of critical constraints violated} / \text{number of constraints violated})$$

The system also distinguishes between two kinds of design constraints: structural and functional and computes each designer's weaknesses with respect to structure and function. Weakness of structural and functional knowledge is represented by the ratios  $R_{\text{WeaknessStructural}}$  and  $R_{\text{WeaknessFunctional}}$ . A high ratio  $R$  indicates that the designer is weak in that category of knowledge.

$$(7.3) \quad R_{\text{WeaknessStructural}} = (\text{number of structural constraints violated} / \text{number of structural constraints relevant to the current design task})$$

---

<sup>9</sup> As the knowledge level is unknown at first, these ratios are initially (nil, nil). The program distinguishes this initial level from the perfect mastery level of (0, 0).

$$(7.4) \quad \mathbf{R}_{\text{WeaknessFunctional}} = (\text{number of functional constraints violated} / \text{number of functional constraints relevant to the current design task})$$

Based on its long-term user model the Furniture Design Critic infers which critiquing methods work well for each designer. When a designer repeatedly succeeds in resolving a constraint violation in response to the critiques offered using a particular method, the program notes that method as an effective critiquing method for that designer.

I could have designed the program to infer designer's abilities and store them in the *User Model*, but instead I decouple the assessments about the designer from the *User Model*, for the sake of the program's generality. The program infers the designer's weaknesses in the structural and functional knowledge, overall knowledge level, and effective critiquing methods. For example, if I add another kind of weakness, say in 'aesthetic knowledge' – aesthetic constraints, the program can take the history of all constraints and compute the designer's aesthetic weakness in the *Pedagogical Module* by defining a new *Critiquing Rule*. I do not need to change the *User Model* to judge the designer's aesthetic weakness. Specifically, if the ratio, (e.g., number of aesthetic constraints violated / number of aesthetic constraints relevant to the current design task) is higher than a certain number while executing a certain *Critiquing Rule*, it indicates that the designer is weak in that aesthetic category of knowledge. This design enables the system designer to scale and generalize this program for supporting a larger design domain or other design domains.

### 7.6.3. Selecting delivery types and communication modalities

Based on the judgments about the designer (critiquing conditions), the *Pedagogical Module* selects delivery types and communication modalities. The *Pedagogical Module* creates a text file that stores a representation of these critiquing conditions. It uses this file to make a decision about which critiquing methods are appropriate in each situation.

Here is an example using the second critiquing scene in Ben's Vignette presented in Sections 5.5.2 and 6.3. The *Pedagogical Module* takes the data of Ben's short-term model and generates the following representation.

```
(short-term-model  
(constraint-ID 37)  
(number-of-violation 1)  
(relevantParts '("2.3.4.1.", (3 4)))  
(usedCritiquingDeliveryTypes '((1 2)))  
(usedCritiquingModalities '((1))))
```

The **constraint-ID** is a unique numerical ID that each constraint has. The value 1 in the slot **number-of-violation** means that the constraint #37 was violated once. The descriptor **relevantParts** indicates what furniture parts are involved in this constraint violation, **'("2.3.4.1.", (3 4))**. Here "2.3.4.1." means the ID of the top part in his table. Vertices **'(3 4)** means that two vertices (corners) of his table do not have supports: V3 and V4. The descriptors **usedCritiquingDeliveryTypes** and **usedCritiquingModalities** means the critiquing methods used when this constraint was previously violated. The value **'(1 2)** in the slot of **usedCritiquingDeliveryTypes** means that 'interpretation' and

‘introduction’ were used together. Also the value ‘(1)’ in the slot of **usedCritiquingModalities** means that ‘written comments’ were used.

The *Pedagogical Module* also takes Ben’s critiquing conditions such as the long-term *User Model* (See Section 7.5) and generates them as follows:

```
(critiquing-conditions  
 (designer "Ben")  
 (historyConstraints '(C0, C1, C2, ....))  
 (states '(S0, S1, S2,....)))
```

Also, the *Pedagogical Module* stores the history of all design constraints and all *States* (see Section 7.10). In the above example, **historyConstraints** '(C0, C1, C2, ....)' means the history of all violated and satisfied constraints. The descriptor **states** '(S0, S1, S2,....)' means all previous *States* such as S1, S2, and S3.

The *Pedagogical Module* computes several ratios for representing Ben’s abilities by using **historyConstraints** while executing the *Critiquing Rules*. Here is a list of ratios that will be used to select a set of critiquing methods.

```
(knowledgeLevel-Rviolated 0.1)  
(knowledgeLevel-RviolatedCritical 0.0)  
(weaknessStructuralCons 0.09)  
(weaknessFunctionalCons 0.12)  
(effectiveCritiquingMethods '())
```

The descriptor **knowledgeLevel-Rviolated** stores Ben's  $R_{Violated}$  ratio, 0.1. The **knowledgeLevel-RviolatedCritical** stores Ben's  $R_{ViolatedCritical}$  ratio, 0.0. The **weaknessStructuralCons** means the ratio,  $R_{WeaknessStructural}$  and the **weaknessFunctionalCons** records the ratio,  $R_{WeaknessFunctional}$ . The descriptor **effectiveCritiquingMethods** stores Ben's effective methods. Here, the value is an empty list, which means that the program has not inferred about his effective critiquing methods yet.

To select an appropriate set of critiquing methods, the *Pedagogical Module* applies *Critiquing Rules* into the critiquing condition representation. It executes actions such as selecting critiquing method candidates and adding a delivery type, if the data of the text file satisfy the condition of a *Critiquing Rule*.

For example, in Ben's above case, Ben's knowledge level satisfies the conditions of the following *Critiquing Rule*: two ratios, **knowledgeLevel-Rviolated** and **knowledgeLevel-RviolatedCritical** are both smaller than thresholds. As his knowledge level satisfies these conditions, the program selects 'interpretation' and 'introduction/ reminder' as delivery type candidate in Step 1 (See Figure 6.6). .

<b>Critiquing Rule#</b>	<b>knowledgeable-designer</b>
<b>[conditions]</b>	if (and (knowledgeLevel-Rviolated $\leq$ 0.2 ) (knowledgeLevel-RviolatedCritical $\leq$ 0.1 ))
<b>[actions]</b>	select delivery type candidates (interpretation + introduction/ reminder)

As Ben's profile satisfies the condition of the following *Critiquing Rule*, another delivery type, 'demonstration' is added in Step 2.

<b>Critiquing Rule#</b>	<b>previously-used-delivery-type</b>
-------------------------	--------------------------------------

<b>[conditions]</b>	if (and (number-of-violation > 0) (usedCritiquingDeliveryTypes contain '(interpretation + introduction/ reminder))
<b>[actions]</b>	add demonstration into the delivery type candidate

This example shows how the *Pedagogical Module* selects an appropriate set of critiquing methods: (1) taking data of the short-term user model and making an inference about the designer using the *User Model*, (2) generating a text file that contains these data, which represent critiquing conditions, and (3) taking the text file and applying *Critiquing Rules* into these data to make a decision about which method is appropriate.

## 7.7. Critiquing Rules

*Critiquing Rules* tell the Furniture Design Critic which delivery types and communication modalities to use under what conditions. The following pseudo-code shows a *Critiquing Rule* that selects delivery type candidates, (demonstration, evaluation, (demonstration, evaluation), when the designer has violated a significant proportion of constraints, indicating that the designer's knowledge level is quite low.

<b>Critiquing Rule#</b>	<b>Delivery-Types-Designer-LowKnowledgeLevel</b>
<b>[conditions]</b>	if ( $R_{Violated} \geq 0.4$ ) and ( $R_{ViolatedCritical} \geq 0.5$ )
<b>[actions]</b>	select delivery type candidates (demonstration, evaluation)

## 7.8. Critic Presenter

Once the *Pedagogical Module* has selected a set of critiquing methods, the *Critic Presenter* activates one or more of its components to present the critique to the designer. The *Critic Presenter* has three

components: A Text Critic presents written comments associated with the violated constraint. A Graphic Critic highlights relevant furniture parts and draws graphical annotations on the designer's diagram. And, an Example Finder retrieves relevant examples from a library and displays them.

So for example if the *Pedagogical Module* selects 'introduction/ reminder' as the delivery type and 'graphical annotation' and 'written comments' as communication modalities, then the *Critic Presenter* activates two components, the Text Critic and the Graphic Critic. The Text Critic takes a stored written message for the violated constraint and presents it. The Graphic Critic executes function calls stored in *critiques-modalities* to annotate the designer's diagram. (It uses the parameters of relevant previously stored furniture examples to position and dimension the annotations.)

Or, if the *Pedagogical Module* selects 'example' as the delivery type and 'images' and 'written comments' as the communication modalities, the *Critic Presenter* activates the Text Critic and the Example Finder. The Example Finder compares saved designs with the current design to retrieve relevant cases. It looks through the saved designs that other designers have made or the current designer has created previously<sup>10</sup>. Suppose the system is critiquing a book case design with a lateral loading problem. The Example Finder scans the previously saved furniture design files. If it finds a bookcase design that satisfies the lateral loading constraint, the Example Finder decides that the saved design is relevant, so it retrieves and presents it. The designer can then learn from the example

---

<sup>10</sup> While saving a furniture design, the program stores three kinds of data in a text file: the parsed data that the *Parser* has generated, the geometrical data from the drawn diagram, and a list of violated and satisfied constraints for the design.

how to resolve the lateral loading issue. The Example Finder also can find counterexamples (i.e., designs that violated the constraint) and present other bookcase designs that had the same issue.

## **7.9. Critiquing Interface**

The program has another interface that is useful to trace its context-sensitive critiquing. This interface is shown in Figure 7.8, Figure 7.9, and Figure 7.11. It graphically presents the program's performance, including the current model that the system has inferred about the user (Figure 7.8), the selected delivery types and communication modalities (Figure 7.9), and the history of all stored states (Figure 7.11). This interface is useful, because it enables us to see the program's performance and reasoning results. It also helps designers understand themselves; for example, they can see their own knowledge levels, weaknesses, and the interaction between them and the program.

The User Model Window (Figure 7.8) graphically displays the results of the program's reasoning about the current user including knowledge level, weakness regarding structural and functional knowledge, and critiquing methods that have proven effective for this user in the past.

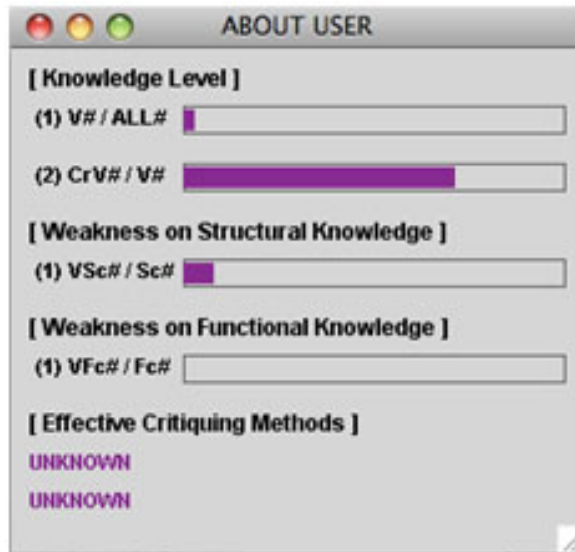


Figure 7.8 A User Model Window displays the inferred data about the current user.

The “Selected Critiquing Methods” window in Figure 7.9 presents the critiquing methods the program has chosen for a particular design state. Each cell in the 5 X 3 table represents one of the five delivery types and one of the three communication modalities. The highlighted cells in Figure 7.9 show that in this state, the system has chosen the delivery type “*introduction/ reminder*” and the modality “*written comments*” and “*graphical annotation*”.

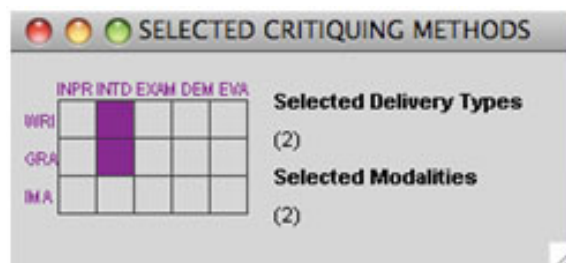


Figure 7.9 Selected delivery types and communication modalities.

## 7.10. States

A *state* represents a critiquing situation: (1) the constraint violated and selected for feedback; (2) the pair of selected delivery types and modalities, (3) the selected delivery types, (4) the selected communication modalities, and (5) whether the selected critiquing methods were effective. The data structure is as follows:

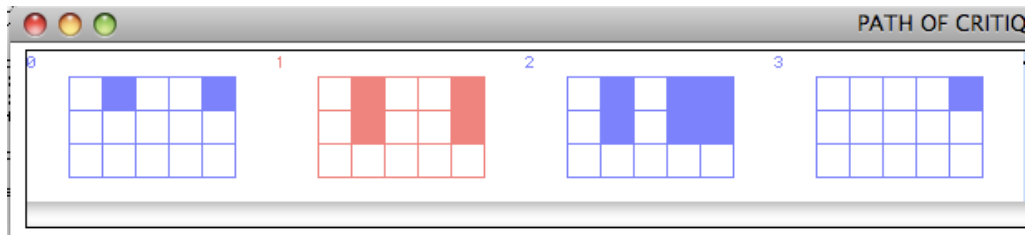
```
(defclass State ()  
  ((state-ID) ; the unique ID of a certain state (numerical id)  
   (violatedCon) ; a constraint violated and selected for feedback  
   (critique) ; a pair of the selected delivery types and modalities  
   (selectedDeliveryTypes) ; the selected delivery types  
   (selectedCommunicationModalities) ; the selected communication modalities  
   (effectiveP))) ; whether the feedback was effective or not
```

This state is captured when the program offers feedback. This is stored as part of the *User Model*. Although it does not explicitly describe characteristics of a designer, the system consults the state when selecting critiquing methods. This history of states may also be useful for a critiquing researcher to examine the history of the design critiquing session. The program displays what critiquing methods it has used and whether the critiques have been successful, which helps to trace the selection of critiquing methods (Figure 7.10 and Figure 7.11).

		<i>Delivery Types</i>				
		Interpretation	Introduction /Reminder	Example	Demonstration	Evaluation
<i>Communication Modalities</i>	Written Comments					
	Graphical Annotations					
	Images					

**Figure 7.10** A state represents the delivery types and communication modalities used at a particular juncture in design critiquing.

The columns in Figure 7.10 represent five different delivery types; the rows represent three communication modalities. Each cell therefore represents a combination of delivery types and communication modalities. A filled cell in a state indicates that, a critique was given using that combination of delivery types and communication modalities. The color of the cell indicates whether the critique was successful—that is, whether it resulted in the designer resolving a constraint violation—blue indicates success; red indicates failure. In Figure 7.10 for example, the critique was delivered as evaluation and was communicated with written comments and graphical annotation. The color (blue) indicates that the selected critiquing methods were successful; the designer resolved the violated constraint.



**Figure 7.11 Path of Critiques:** what critiquing methods were used and whether they were effective

The “Path of Critiques” window in Figure 7.11 shows four successive states in a design history. Each cell in this window represents a state of the design: Earlier states are shown to the left, and more recent ones to the right. In *State 0* (the leftmost grid), the program offered feedback using [evaluation + introduction/ reminder] and [written comments]. The designer, critiqued in this form, successfully resolved the issue the Furniture Design Critic raised. Another constraint was violated in *State 1*. In this state, the program offered feedback using [evaluation + introduction/ reminder] and [written comments + graphical annotations]. However, the designer failed to resolve the raised issue. Then in *State 2*, the program selected an additional delivery type, demonstration, to present the designer with a possible solution; [introduction/ reminder + demonstration + evaluation] and [written comments + graphical annotation]. The blue color shows that the critique was successful. In *State 4*, the program found another constraint violation and provided feedback using [evaluation] and [written comments]. Given this critique, the designer improved the design.

This history of states reveals the trace of feedback that the program offered to a designer. The program also uses the history to select critiquing methods. For example, when a particular set of

methods have been used repeatedly in two previous states and the critiques have been unsuccessful both times, the program will select alternative critiquing methods to offer feedback.

## **8. Conclusions, Contributions, and Future Work**

### **8.1. Summary**

This dissertation presents the Furniture Design Critic program, a computational model of design critiquing. The model views critiquing as selecting a method of critiquing in consideration of critiquing conditions. In order to develop this model, I first developed a theoretical framework by surveying the literature of design studio education. This framework identifies eleven fundamental factors including six critiquing conditions (e.g., design phases, students' individual differences) and five critiquing methods (e.g., delivery types and communication modalities). It provides a conceptual map of the relevant factors, but the framework alone cannot account for how these factors are operationalized.

Based on this framework I implemented a computational model of decision-making in critiquing, the Furniture Design Critic. The program, intended as a research tool, employs the architecture of “constraint-based intelligent tutors”. The program’s *User Model* and *Pedagogical Module* can represent critiquing conditions and methods, and it can be used to investigate and articulate the selection mechanisms of design critiquing. The resulting critiques are context-sensitive and multi-modal.

Using the program, I demonstrated the computational model’s capacity to account for alternative design critiquing strategies. This dissertation describes three hypothetical critiquing sessions, or vignettes. The vignettes illustrate what the Furniture Design Critic program can do; and they show how a critiquing researcher might use the program to postulate different mechanisms for selecting critiquing methods. Each of the three mechanisms shows how the program chooses a particular set of delivery types and communication methods. Each implements a different strategy for critiquing based on the relationships among several conditions.

## **8.2. Contributions**

The major contribution of this research is embodied in a computer-based critiquing system, the Furniture Design Critic. This section summarizes contributions that derive from the development of the program and the examination of the concrete critiquing session examples described in Chapters 6 and 7. Recall that a central goal of this research is to systematically describe design critiquing, and more precisely to account for critiquing strategies as selecting critiquing methods in reference to critiquing conditions. Unlike much of the previous work in computer-based critiquing, the research

presented here is not concerned with how to represent and convey domain knowledge. Rather, it is concerned with finding appropriate ways (i.e., delivery types and communication modalities) to present critiques to designers.

First, the computational model of design critiquing underlying the Furniture Design Critic can be used to investigate various strategies for critiquing. The model describes relationships between two factors of critiquing: the conditions and methods. The program makes inferences about critiquing conditions using its *User Model* and then selects critiquing methods using its *Pedagogical Module* and *Critiquing Rules*. The three vignettes presented in Chapter 5 show that such strategies can be articulated explicitly. Other strategies, in addition to those illustrated in the three vignettes, can be devised, described, and deployed using the Furniture Design Critic program. By adding and modifying *Critiquing Rules* that determine how the program selects critiquing methods a researcher can devise other critiquing strategies for the program to execute.

Second, this work provides a foundation for empirical research on design critiquing, such as case studies. As the literature survey on critiquing in design studio revealed, the field has suffered from a lack of clear definitions, making it difficult to posit testable hypotheses, analyze actual critiquing cases, and interpret empirical findings. The computational model presented here offers definitions of critiquing variables such as knowledge level, weaknesses, delivery types, and communication modalities. And the Furniture Design Critic program provides a means to test hypotheses such as ‘*a designer with a high knowledge level will benefit more from facilitative delivery types than from directive ones.*’

To plan an empirical study about the pedagogical effectiveness of critiquing methods the model can help formulate a clear set of variables and testable hypotheses. To test the hypothesis, ‘*a designer with a high knowledge level will benefit more from facilitative delivery types than from directive ones*,’ we might select designers with a relatively high knowledge level and compare the results of facilitative versus directive feedback. This program also provides a tool to interpret and understand empirical findings and account for a variety of real critiquing sessions. Just as I have described three hypothetical critiquing strategies, we could also describe actual critiquing sessions observed in empirical research.

Third, the Furniture Design Critic program contributes to the field of computer-based critiquing and intelligent tutoring systems. The review of these systems in Chapter 2 found no critiquing or intelligent tutoring system that accounts for strategic decisions about how to present critiques or what is termed here, “delivery type” and “communication modality”. The Furniture Design Critic program and its underlying computational model is the first to account for this strategic decision-making about how to present critiques.

### **8.3. Future Work**

#### **8.3.1. Application in Studio**

Studio teachers can use this critiquing model to develop pedagogically effective critiquing strategies. The model can help teachers understand the complexity of design critiquing, and consider how they might select a critiquing method for various conditions, and so evaluate and improve their critiquing practice.

Although it is often assumed that professional architects are effective teachers, this is not always so. Although architects can *do* design, they may be less skilled at *teaching* it. Studio teachers in general, despite their experience teaching design, lack explicit awareness of design pedagogy. They draw mainly on their own professional education; and so their critiquing methods reflect the way they themselves were taught. They may not reflect on why they critique students the way they do.

Design studio learning encompasses a rich mixture of visual, verbal, graphical, and written forms of representation and communication. Therefore, critiquing in design studios inevitably involves a diverse array of communication strategies; however, seldom do teachers conduct a systematic development and assessment of their communication strategies. Teachers must communicate ideas under different critiquing conditions. Yet at best studio teachers' experience in adjusting their communication to these different conditions is *ad-hoc*.

A future study would investigate whether the computational model of design critiquing can help teachers' critiquing and improve studio teaching practice. The study would include several steps. I would (1) introduce the critiquing model to studio teachers; (2) lead them to propose critiquing strategies and then using these strategies tailor their critiques; (3) let them offer feedback in critiquing sessions; (4) determine whether this feedback actually helps to students learn; (5) describe and articulate why (or why not) certain critiquing strategies work to support students' learning; and thereby (6) lead them to improve their critiquing strategies.

The broader goal is to make studio teachers more reflective about design critiquing and to help teachers develop mechanisms for choosing appropriate critiquing methods.

### **8.3.2. Professional Development of Studio Teachers**

Building on the study proposed in the previous section, the model could also serve as a basis for a studio teacher training program. This would have several aims: (1) to integrate this model with design pedagogy also to be developed in this training program; (2) to test whether the model can support the professional development of studio teachers; and (3) to develop a program that enables future studio teachers to understand design critiquing, to choose and to reflect on appropriate critiquing methods, with a view to refining their critiquing and teaching. I plan a training program for studio teachers in which trainers and studio teacher trainees will wrestle with the question, 'how should we critique design students?' By providing ways to explain and articulate design critiquing, I intend to help studio teachers to develop a systematic understanding of design critiquing.

Typically, learning science experts do not know about designing and professional architects do not think about pedagogy. A team of trainers including a design critiquing researcher, professional architects, and a learning scientist would work together to train studio teachers. Together this team can provide expertise to help teachers learn to craft appropriate critiques. This would test whether the critiquing model presented here is a useful foundation for training.

Trainees would first learn to recognize what they do not know about offering feedback. Trainers can help trainees understand design critiquing; to recognize various critiquing conditions and the advantages, disadvantages, and pedagogical effects of various critiquing methods. Based on these lessons, each trainee will develop a proposal for his or her own critiquing. The proposal will then be discussed with the trainee group and the trainer team in a seminar. The proposal will be examined with the analysis and rationale of the given situations. The proposal for critiquing is then

presented and discussed among the whole group. In these discussions, trainers provide different expert knowledge to help trainees develop and improve their critiquing proposals.

I anticipate four outcomes: (1) the trainees become better prepared to teach and critique students; (2) the trainer group develops a better studio teacher training program; (3) trainers and trainees explore and develop design pedagogy; and (4) the trainer group provides understanding of design critiquing closely integrated with design pedagogy.

### **8.3.3. Expanding the Furniture Design Critic**

The Furniture Design Critic program presented here does not account for all the eleven critiquing factors identified from the theoretical framework of design critiquing. It focuses only on two factors: critiquing delivery types and communication modalities. Future research can extend the system by adding other factors to achieve a more complete model of design critiquing.

The Furniture Design Critic program can be extended in several directions. First, other communication modalities can be added, for example, oral (spoken) feedback. Spoken critiques enable the investigation of another critiquing factor: voice quality – intonation and loudness. Second, the program can make inferences about other critiquing conditions, such as designers' response types (see Chapter 3). If the program understands and records designers' reactions to feedback it has offered, then it can assess their response types. Third, by adding critiquing conditions and methods I can develop and explore other selection mechanisms.

#### 8.3.4. Test Effective Critiquing Methods

The Furniture Design Critic program can be used to investigate the appropriate application of critiquing methods in specific situations: which methods produce the most pedagogically effective response from designers. Although I did not perform this empirical experiment, the Furniture Design Critic computational model offers a basis for formulating hypotheses, for example, the hypothesis that [‘demonstration’ and ‘written comments’ + ‘graphical annotations’] are effective communication methods for a designer who has little design knowledge and low spatial ability.

#### 8.3.5. Extension to Other Domains

The framework, computational model, and program described here was developed and demonstrated in the simple domain of flat-pack furniture. However, this work is not restricted to flat-pack furniture. It can be extended to other domains, for example graphic, architectural, or industrial design. Critiquing can be modeled using the system presented here because the fundamental components are domain independent. To support other design domains, three main components of the Furniture Design Critic — the *User Model*, the *Pedagogical Module* and *Critiquing Rules* — need not be changed.

The *User Model* represents a user based on the history of all *Design Constraints* satisfied and violated. From this history the program assesses various characteristics of the user. It assesses knowledge level, effective critiquing methods and two categories of user weakness (structural and functional knowledge). The *User Model* can be extended to assess other characteristics of users for supporting other domains.

The *Pedagogical Module* is also domain independent. This module takes a short-term user model and the long-term user model (*User Model*) and selects a set of critiquing methods by applying the *Critiquing Rules*. This module need not be changed to support other domains.

Although the *Critiquing Rules* in the current version of the Furniture Design Critic should not necessarily be considered prescriptively correct, these rules also are domain independent. Each *Critiquing Rule* takes the *User Model* and selects a critiquing method that is appropriate for that set of critiquing conditions. These critiquing conditions and methods are domain independent, so that *Critiquing Rules* need not be changed for other domains.

## 8.4. Conclusion

Design critiquing, as it is practiced in design studio by skilled teachers, is a subtle art. It will be some time before a computer program can provide as valuable feedback to design students as a human design critic. The first computer-based critiquing programs were limited to pointing out simple errors; later generations of critiquing software have provided increasingly subtle and sophisticated design advice. Whereas earlier generations of critiquing software communicated through text alone, more recent software uses graphic annotation and pictures as well as text.

At the same time, through learning science research we are becoming aware of the differences in individual learning styles that make some critiquing strategies more effective for some learners, and other strategies more effective for others. We are starting to understand what we do not know about critiquing strategies may limit the effectiveness of human studio critics. This missing knowledge is required for building effective computer based critics.

This dissertation has drawn on several sources to investigate computer based design critiquing. The literature of design studio teaching provided a framework for a computational model. Previous efforts to build computer based critics have informed the computational model, as has work on constraint-based intelligent tutoring systems. The Furniture Design Critic's computational model offers a way to account for aspects of design critiquing that previous systems have not included—what is termed here “delivery type” and “communication modality”. Both these aspects can have profound effects on the effectiveness of critiquing in studio, so there is every reason to believe that future critiquing systems should incorporate strategies to control these variables.

## References

- Ahrentzen, S. and K. H. Anthony (1993). "Sex, Stars, and Studios: A Look at Gendered Educational Practices in Architecture." Journal of Architectural Education **47**(1): 11 - 29.
- Aleven, V. and K. R. Koedinger (2000). Limitations of Student Control: Do Students Know When They Need Help. 5th International Conference on Intelligent Tutoring Systems (ITS), Montreal, Springer: 292 - 303.
- Anderson, J. R., A. T. Corbett, K. R. Koedinger, et al. (1995). "Cognitive Tutors: Lessons Learned." The Journal of the Learning Sciences **4**(2): 167 - 207.
- Anderson, J. R., R. Farrell and R. Sauers (1984). "Learning to Program in Lisp." Cognitive Science **8**(2): 87 - 130.
- Anthony, K. H. (1991). Design Juries on Trial: The Renaissance of the Design Studio. New York, Van Nostrand Reinhold.
- Argyris, C. (1981). Teaching and Learning in Design Settings Architecture Education Study. New York, Andrew W. Mellon Foundation: 551 - 660
- Atman, C. J., J. R. Chimka, K. M. Bursic, et al. (1999). "A Comparison of Freshman and Senior Engineering Design Processes." Design Studies **20**(2): 131-152
- Attoe, W. and R. Mugerauer (1991). "Excellent Studio Teaching in Architecture " Studies in Higher Education **16**(1): 41 - 50

- Bailey, R. O. N. (2004). The Digital Design Coach: Enhancing Design Conversations in Architecture Education. Architecture Victoria University of Wellington **PhD**.
- Boyer, E. L. and L. D. Mitgang (1996). Building Community: A New Future for Architecture Education and Practice.
- Chun, H. W. and E. Ming-Kit Lai (1997). "Intelligent Critic System for Architectural Design." IEEE Transactions on Knowledge and Data Engineering **9**(4): 625 - 639.
- Corbett, A. T., H. J. Trask, K. C. Scarpinato, et al. (1998). A Formative Evaluation of the PACT Algebra II Tutor: Support for Simple Hierarchical Reasoning. the 4th International Conference on Intelligent Tutoring Systems (ITS), San Antonio, Texas, USA, Springer-Verlag: 374 - 383.
- CORENET. (2004). "CORENET e-Plan Check System." 2007, from <http://www.corenet.gov.sg/corenet/>.
- Cuff, D. (1991). Architecture: the Story of Practice, The MIT Press
- Dinham, S. M. (1986). "Architecture Education: Is Jury Criticism a Valid Teaching Technique?" Architecture Record: 51 - 53
- Dutton, T. A. (1991). The Hidden Curriculum and the Design Studio Voices in Architectural Education: Cultural Politics and Pedagogy T. A. Dutton. New York, Bergin and Gravey: 165 - 194.
- Farivarsadri, G. (2001 ). A Critical View of Pedagogical Dimension of Introductory Design in Architectural Education Architectural Education Exchange 2001 Cardiff University
- Fischer, G. (1987). A Critic for LISP. The 10th International Joint Conference on Artificial Intelligence, Milan, Italy: 177 - 184.
- Fischer, G. (1989). "Human-Computer Interaction Software: Lessons Learned, Challenges Ahead." IEEE Software **6**(1): 44 - 52.
- Fischer, G., A. C. Lemke, T. Mastaglio, et al. (1991). "The Role of Critiquing in Cooperative Problem Solving." ACM Transactions on Information Systems **9**(2): 123 - 151.
- Fischer, G., A. C. Lemke, R. McCall, et al. (1991). "Making Argumentation Serve Design." Human Computer Interactions **6**(3 - 4): 393 - 419.

- Fischer, G., R. McCall and A. Morch (1989). JANUS: Integrating Hypertext with a Knowledge-Based Design Environment. ACM Conference on Hypertext and Hypermedia, Pittsburgh, PA, ACM Press: 105 - 117.
- Fischer, G., R. McCall and A. I. Morch (1989). Design Environments for Constructive and Argumentative Design. Human Factors in Computing Systems (CHI '89), Austin, Texas, ACM Press: 269 - 275.
- Fischer, G. and A. I. Morch (1988). Crack: A Critiquing Approach to Cooperative Kitchen Design. ACM International Conference on Intelligent Tutoring Systems, Montreal, Canada, ACM Press: 176 - 185.
- Fischer, G., K. Nakakoji, J. Ostwald, et al. (1998). Embedding Critics in Design Environments. Readings in Intelligent User Interfaces. San Francisco, CA, USA, Morgan Kaufmann Publishers Inc.: 537 - 561.
- Fu, M. C., C. C. Hayes and E. W. East (1997). "SEDAR: Expert Critiquing System for Flat and Low-slope Roof Design and Review." Journal of Computing in Civil Engineering **11**(1): 60 - 68.
- Gertner, A. S. and B. L. Webber (1998). "TraumaTIQ: online decision support for trauma management." IEEE Intelligent Systems **13**(1): 32 - 39.
- Goldschmidt, G. (2002). One-on-One: A Pedagogic Base for Design Instruction in the Studio. Common Ground Design Research Society International Conference Brunel University Staffordshire University Press 430 - 437
- Goldschmidt, G. (2003). Expert Knowledge or Creative Spark? Predicaments in Design Education. Design Thinking Research Symposium 6, University of Technology, Sydney, Australia.
- Grasha, A. F. (1996 ). Teaching with Style: a practical guide to enhancing learning by understanding teaching and learning styles. Pittsburgh, Alliance Publishers.
- Groat, L. and S. Ahrentzen (1996). "Reconceptualizing Architectural Education for a More Diverse Future: Perceptions and Visions of Architectural Students." Journal of Architecture Education **49**(3): 166 - 183.
- Han, C. S., K. H. Law, J.-C. Latombe, et al. (2002). "A Performance-Based Approach to Wheelchair Accessible Route Analysis." Advanced Engineering Informatics **16**(1): 53-71.

- Heffernan, N. T., K. R. Koedinger and L. Razzaq (2008). "Expanding the Model-Tracing Architecture: A 3rd Generation Intelligent Tutor for Algebra Symbolization." The International Journal of Artificial Intelligence in Education **18**(2): 153 - 178.
- Holt, P., S. Dubs, M. Jones, et al. (1994). The State of Student Modelling. Student Modelling: The Key to Individualized Knowledge-based Instruction. J. E. Greer and G. I. McCalla. Berlin, Springer-Verlag: 3 - 35.
- Kent, L. A. (2001). The Case of Lucio Pozzi: An Artist/Teacher's Studio Critique Method. Teachers College. New York, Columbia University. **PhD**: 354.
- Koch, A., K. Schwenksen, T. A. Dutton, et al. (2002). The Redesign of Studio Culture: A Report of the AIAS Studio Culture Task Force, The American Institute of Architecture Students
- Koedinger, K. R., V. Aleven, N. Heffernan, et al. (2004). Opening the Door to Non-programmers: Authoring Intelligent Tutor Behavior by Demonstration. Intelligent Tutoring Systems (ITS), Maceio, Brazil, Springer: 162 - 174.
- Kolb, D. (1984). Experiential learning. Englewood Cliffs, NJ, Prentice-Hall.
- Martin, B. and A. Mitrovic (2002). "WETAS: A Web-based Authoring System for Constraint-based ITS." Lecture Notes in Computer Science **2347**: 543 - 546.
- Mastaglio, T. (1990). User Modeling in Computer-based Critics. IEEE The 23rd Annual Hawaii International Conference on System Sciences, Kailua-Kona, HI, USA, IEEE Press: 403 - 412.
- Mayer, R. E. (2001). Multimedia Learning, Cambridge University Press.
- Mayo, M., A. Mitrovic and J. McKenzie (2000). Capit: An Intelligent Tutoring System for Capitalisation and Punctuation. International Workshop on Advanced Learning Technologies 2000 (IWALT 2000), Palmerston North, New Zealand: 151 - 154.
- Milik, N., M. Marshall and A. Mitrovic (2006). "Responding to Free-form Student Questions in ERM-Tutor." Lecture Notes in Computer Science **4053**: 707 - 709.
- Mitrovic, A. (1997). SQL-Tutor: A Preliminary Report, Computer Science Department, University of Canterbury.
- Mitrovic, A. (2002). NORMIT: A Web-Enabled Tutor for Database Normalization. International Conference on Computers in Education (ICCE): 1276 - 1280.

- Mitrovic, A., B. Martin and P. Suraweera (2007). "Intelligent Tutors for All: The Constraint-Based Approach." IEEE Intelligent Systems **22**(4): 38 - 45.
- Nakakoji, K., Y. Yamamoto, T. Suzuki, et al. (1998). "From Critiquing to Representational Talkback: computer support for revealing features in design." Knowledge-Based Systems **11**(7-8): 457 - 468.
- Odgers, J. (2001). Authority, Questioning and Learning: Reflections on writing as reflective practice in the design studio Architectural Education Exchange 2001 Cardiff University.
- Oh, Y., E. Y.-L. Do and M. D. Gross (2004). Intelligent Critiquing of Design Sketches. AAAI (American Association for Artificial Intelligence) Fall Symposium - Making Pen-based Interaction Intelligent and Natural, Washington DC, The AAAI Press: 127 - 133.
- Oh, Y., G. Johnson, M. D. Gross, et al. (2006). The Designosaur and the Furniture Factory: simple software for fast fabrication. International Conference on Design Computing and Cognition (DCC 2006), Eindhoven, the Netherlands, Springer: 123 - 140.
- Ohlsson, S. (1994). Constraint-based Student Modeling. Student Modelling: The Key to Individualized Knowledge-based Instruction. J. E. Greer and G. I. McCalla. Berlin, Springer-Verlag: 167 - 189.
- Ohlsson, S. (1996). "Learning from Performance Errors." Psychological Review **3**(2): 241 - 262.
- Parnell, R. (2001). It's Good to Talk: Managing Disjunction through Peer Discussion Architectural Education Exchange 2001 Cardiff University
- Qiu, L. and C. K. Riesbeck (2004). Incremental Authoring of Computer-based Interactive Learning Environments for Problem-based Learning. IEEE International Conference on Advanced Learning Technologies (ICALT), Finland, IEEE Press: 171 - 175.
- Rich, E. (1989). Stereotypes and User modeling. User Models in Dialog Systems. A. Kobsa and W. Wahlster. Berlin, Springer: 35 - 51.
- Roach, P., R. Stibbard, J. Osborne, et al. (1998). "Transcription of Prosodic and Paralinguistic Features of Emotional Speech." Journal of the International Phonetic Association **28**: 83 - 94.
- Robbins, J. E. (1998). Design Critiquing Systems. Tech Report UCI-98-41, Department of Information and Computer Science, University of California, Irvine.

- Robbins, J. E. and D. F. Redmiles (1998). "Software Architecture Critics in the Argo Design Environment." Knowledge-Based Systems **11**(1): 47 - 60.
- Rohrbach, S. (2005). Responding to the Changing Roles of Designers: Fostering the Critical Analysis and Verbal Communication Skills of Students through the Evaluation of Design Work. AIGA National Design Education Conference, Philadelphia, PA, USA.
- Schön, D. A. (1983). The Reflective Practitioner: How Professionals Think in Action Basic Books Inc. .
- Schön, D. A. (1985). The Design Studio. London, RIBA.
- Silverman, B. G. (1992). Critiquing Human Error: A Knowledge Based Human-Computer Collaboration Approach, Academic Press.
- Silverman, B. G. (1992). "Survey of Expert Critiquing Systems: Practical and Theoretical Frontiers." Communications of the ACM **35**(4): 106 - 127.
- Solibri.Inc. (2007, 09/13/2007). "Solibri Inc. The World Leader in Design Spell Checking." Retrieved 08/12/2007, 2007, from <http://www.solibri.com/>.
- Souza, C. R. B. d., H. L. R. Oliveira, C. R. P. d. Rocha, et al. (2003). Using Critiquing Systems for Inconsistency Detection in Software Engineering Models. The Fifteenth International Conference on Software Engineering and Knowledge Engineering (SEKE 2003), San Francisco, CA, USA: 196 - 203.
- Straub, R. (1996). "The Concept of Control in Teacher Response: Defining the Varieties of "Directive" and "Facilitative" Commentary." College Composition and Communication **47**(2): 223 - 251.
- Suraweera, P. and A. Mitrovic (2002). "Kermit: A Constraint-Based Tutor for Database Modeling." Intelligent Tutoring Systems **2363**: 377 - 387.
- Torrey, C. (2009). How Robots Can Help: Communication Strategies that Improve Social Outcomes. Human Computer Interaction Institute. Pittsburgh, PA, Carnegie Mellon University. **PhD.**
- Uluoglu, B. (2000). "Design Knowledge Communicated in Studio Critiques " Design Studies **21**(1): 33 - 58
- Ulusoy, Z. (1999). "To Design versus to Understand Design: the Role of Graphic Representations and Verbal Expressions " Design Studies **20**(2): 123 - 130

- Wampler, J. (2002). Architecture Studio: Building in Landscapes, MIT Open Courseware.
- Weaver, N., D. O'Reilly and M. Caddick (2000). Preparation and Support of Part-Time Teachers: Designing a Tutor Training Programme Fit for Architects. Changing Architectural Education: Towards a New Professionalism. D. Nicol and S. Pilling. New York, Taylor & Francis Spon Press: 265 - 273.
- Willenbrock, L. L. (1991). An Undergraduate Voice in Architectural Education Voices in Architectural Education: Cultural Politics and Pedagogy T. A. Dutton, New York: Bergin and Gravey pp 97 - 120.
- Ye, Y. (2003). "Programming with an Intelligent Agent." IEEE Intelligent Systems **18**(3): 43 - 47.
- Young, D. E. (2007, 2007. July). "The Lisa Project." Retrieved Oct. 16, 2008, from <http://lisa.sourceforge.net/>.