ClassCompass: A Software Design Mentoring System

WESLEY COELHO and GAIL MURPHY
University of British Columbia

Becoming a quality software developer requires practice under the guidance of an expert mentor. Unfortunately, in most academic environments, there are not enough experts to provide any significant design mentoring for software engineering students. To address this problem, we present a collaborative software design tool intended to maximize an instructor's ability to mentor a group of students. Students use the system to create software designs for a given set of requirements. While they work, students receive automated feedback regarding common design mistakes. The system then provides support and guidance for students to manually critique each other's work. Students can view and learn from the design approaches taken by other students, as well as the critiques associated with them. We have tried this approach in software engineering classes with some positive results. We believe that this collaborative and partially automated approach can significantly improve the quality of software design education when few mentors are available.

Categories and Subject Descriptors: D.2.2 [**Software Engineering**]: Design Tools and Techniques; D.1.5 [**Programming Techniques**]: Object-oriented Programming

General Terms: Design

Additional Key Words and Phrases: Collaborative education, design critiquing

ACM Reference Format:

Coelho, W. and Murphy, G. 2007. ClassCompass: A software design mentoring system. ACM J. Educ. Resour. Comput. 7, 1, Article 2 (March 2007), 18 pages. DOI = 10.1145/1227846.1227848 http://doi.acm.org/10. 1145.1227846.1227848

1. INTRODUCTION

Becoming a quality software developer requires both the acquisition of theory and the practice of skills under the guidance of an expert. It is through the act of apprenticing with an expert that a student learns how the theory applies to practice. Although apprenticeship is accepted in many areas, including traditional professionally-based engineering and medicine, no similar notion of apprenticeship is in place for computer science and engineering students to

Author's address: G. Murphy, Department of Computer Science, University of British Columbia, 201-2366 Main Mall, Vancouver BC, Canada, V6T 1Z4; email: murphy@cs.ubc.ca.

become software developers. As a result, both academia and industry bemoan the level of development expertise exhibited by students [Lethbridge 1998]. Although faculty members in post-secondary institutions work hard to provide their students with a solid software engineering education, there are simply too few of them to act as both transmitters of theory and mentors to apprentices. In addition to the lack of mentors, software development education is further hampered by the limited effectiveness of current teaching practices. Instructors typically give lectures describing software design principles[1] and present classic examples of how they are used. Students read textbook accounts of design issues and complete design assignments. These design assignments are often graded and returned with little feedback long after they were submitted. Since there is no immediate feedback and no iterative review and revision, students have little opportunity to improve their skills.

To address these education concerns, we have developed a distributed tool for software design mentoring called ClassCompass[2]. ClassCompass provides students with a convenient graphical editor for creating and revising software designs specified using UML class diagrams[3]. As students work on their designs, an automatic critique system provides advice when common mistakes are made that could reduce software quality. After initial designs are completed, the ClassCompass system provides tools and guidance for students to critique each other's designs. An expert mentor may view designs and critiques to provide additional feedback. When the design and critique process is complete, students can learn from the different designs and corresponding critiques that were produced.

In this article, we describe three contributions made by this work. We introduce an automated software design critique system with critics that comment on high-level design issues rather than diagram completeness. We describe how to support guided manual critiquing of software designs. We also describe a distributed tool that uses the automated and manual critique features to support group design exercises led by a mentor as well as collaboration among course-project group members.

We present scenarios of how ClassCompass is used as an education tool in Section 2. Section 3 provides an overview of ClassCompass describing the infrastructure, automated critiquing system, support for manual critiquing, and instructional session administration. In Sections 4 and 5, we provide an evaluation and discussion of ClassCompass. Related work is discussed in Section 6, and we conclude in Section 7.

2. DESIGN EDUCATION SCENARIOS

There are several ways to use ClassCompass as an education tool for software design. The student-client application can be used as a standalone tool to

[1]Examples of design principles can be found in Page-Jones and Constantine [1999], Liskov and Wing [1994], and Lieberherr et al. [1988].

[2]The ClassCompass tool can be downloaded from `http://www.cs.ubc.ca/labs/spl/projects/classcompass`.

[3]The approach could also be applied to support other UML diagrams such as sequence diagrams.

practice software design. When a network connection is available, greater benefit can be obtained from using the system in collaboration with other students. We present two scenarios where ClassCompass can be used for collaborative learning in a software engineering course or in the workplace.

2.1 Scenario 1: In-Class Exercise

An instructor is planning a class on software design principles where students will practice the principles by creating software designs and then critiquing the designs of other students. Before the class, the instructor configures a session (as described in Section 3.1) to support the design principles that will be covered, such as low coupling, and textually describes a software design problem.

During the class, the instructor presents several design principles and then poses the prepared software design problem. For example, the students might be asked to produce a design for a pay-at-the-pump system for a gas station as described below.

> A pay-at-the-pump system is required to operate one or more fuel pumps as well as the user interface hardware for processing payments. The system will allow customers to insert a credit card and select a grade of fuel. The customer will then pump fuel using either a gasoline or diesel variant of the pump from the same manufacturer. However, the system must also support a gasoline pump from another manufacturer that uses entirely different communication protocols. When the user is finished pumping, the system must retrieve the volume pumped from the pump that was used and compute the total sale. The sale amount and volume of fuel pumped will be displayed on an LCD screen and printed on a receipt. Please describe a software design for this system.

The students break into small groups with at least one Internet-enabled laptop available per group. The students run ClassCompass, connect to the previously prepared instructional session, and begin sketching a UML class diagram that would provide a solution to the design problem. As the students work, the automated design critiquing system within ClassCompass identifies parts of the design that may violate well-known design principles or typical design conventions. For example, the system may point out parts of the class diagram where there are multiple association paths between classes, suggesting that the classes are more coupled than is needed. The students working on that design may choose whether or not to accept the advice and update the design. Figure 1 shows the ClassCompass user interface when an automatic critique is being presented. Automatic critiques are discussed in more detail in Section 3.3.

After a few minutes, the instructor asks the students to submit their designs. The instructor then uses the ClassCompass client (instructor version) to automatically exchange designs between groups of students. Students begin to critique each other's designs based on the principles previously reviewed. To make these critiques, students use the tools provided by ClassCompass.

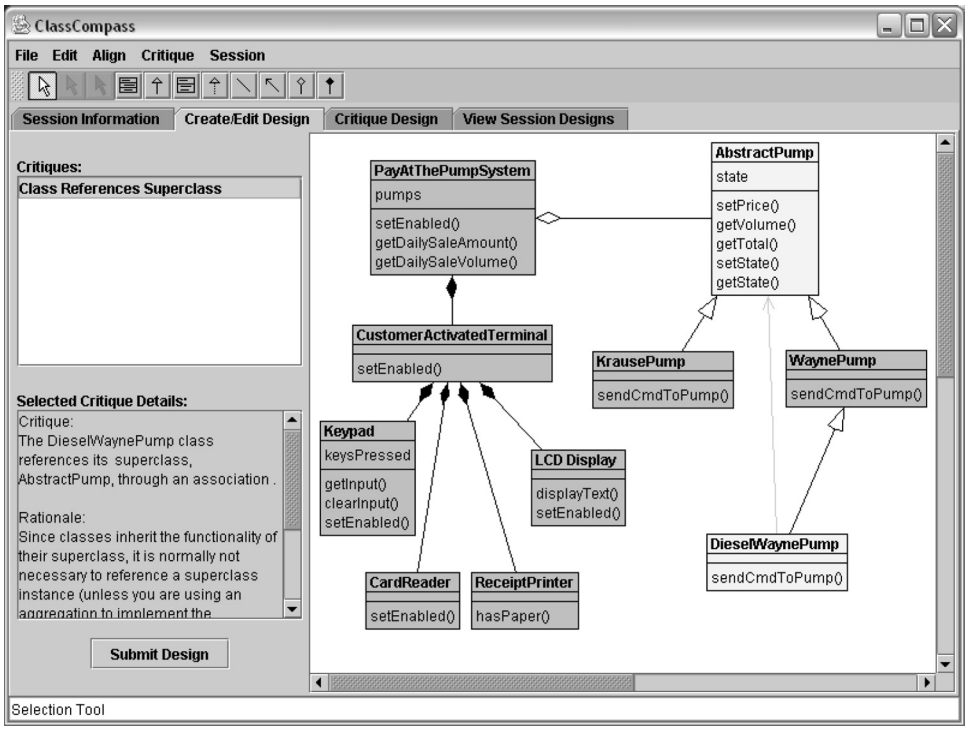

Fig. 1. The student client program while editing a design. The user has requested more information about an automatic critique. The details of the critique appear in the lower left and the relevant elements of the diagram have been highlighted.

When all designs and critiques have been submitted, the instructor may use them to motivate a subsequent discussion. During and after the workshop, students are able to benefit from viewing other students' designs as well as the critiques that were submitted for them. Figure 2 provides an overview of the sequences of activities that are supported by ClassCompass for in-class exercises.

2.2 Scenario 2: Group Project

Several groups of students are required to produce designs for a small software system as part of a software engineering course project. To complete the project, the group members need to work collaboratively on the software design.

A private ClassCompass session has been established by the course instructor for each group to use. Students in a group connect to the session using ClassCompass and begin creating the design. The students use the automated critiques to gain immediate feedback on their initial design and submit it to the ClassCompass server. Subsequently, individual team members may log into the ClassCompass session from home or a computer lab to make design changes or refinements as the project progresses. Other team members can log into the session and use the manual critique features to critique the latest version

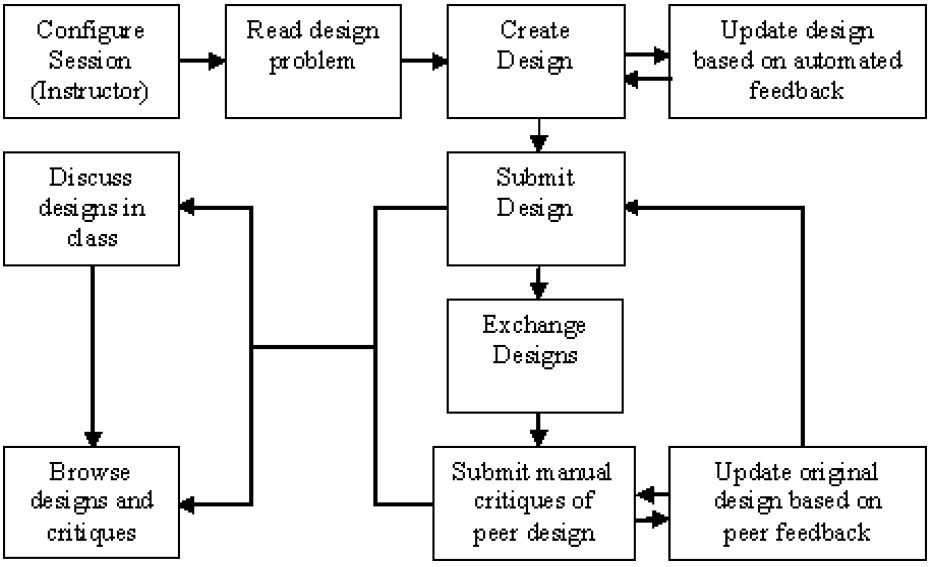

Fig. 2. Flow of activities supported by ClassCompass for in-class exercises.

of the design. Revised versions of the design are submitted in response to the manual critiques. Several iterations of designing and critiquing help to produce a quality design for the project.

3. CLASSCOMPASS

The ClassCompass design-mentoring system consists of several components. Instructors use a Web browser to configure the system for instructional sessions accessed by participating students. These instructional sessions are hosted by a session server, which stores session state such as design problems, diagrams, and critiques in a relational database. Students join and participate in shared sessions hosted by the session server using a Java client. The student-client program provides functionality for creating, viewing and critiquing software designs. The student client can also be used as a standalone tool for automated design critiquing. Instructors use an extended version of the student client that provides additional features for managing a session. The main components of the ClassCompass system are illustrated in Figure 3. In this architecture, a single server can be made available to a number of institutions. Each institution will then be able to use the system simply by downloading the Java client and configuring sessions with a Web browser.

3.1 Instructional Session Administration

Instructors use a Web application to configure ClassCompass before students engage in collaborative design exercises. This Web application provides several features for creating and maintaining the instructional sessions that are hosted by the ClassCompass server. Students connect to these sessions in order to view design problems and share designs and critiques.

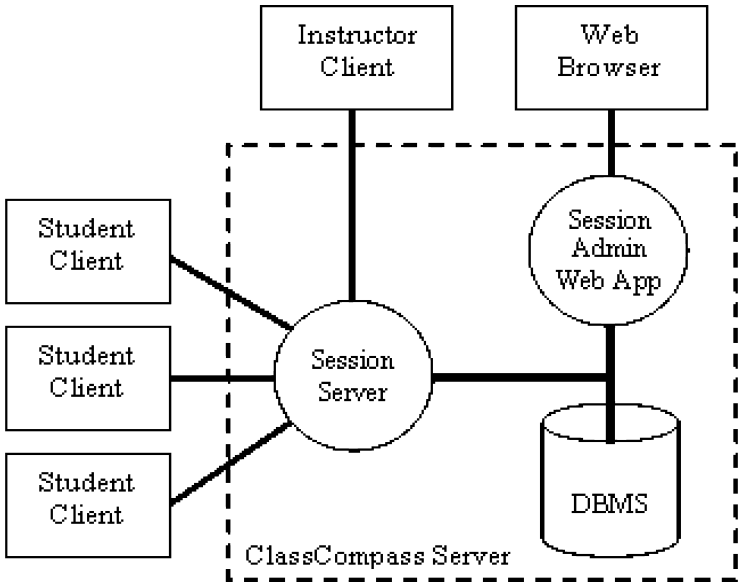

Fig. 3. Main components of the ClassCompass system.

Table I. An Example of Design Principle Information Specified by an Instructor Using the Session Administration System

Title	Low Coupling
Description	The number of interactions between classes should be minimized
Instructions	Use the red structure selection tool to select classes and links that are more coupled than you think necessary. Explain how the links you identified could be removed.

The session administration web application allows instructors to specify the design principles that students use to manually evaluate designs. A design principle consists of a title, brief description, and evaluation instructions. Evaluation instructions indicate how the structure selection tools should be used as well as what is expected in the critique text area. Table I provides an example of design principle information entered by an instructor.

When one or more design principles have been entered, instructors can create instructional sessions to which students may connect. When creating a session, a description of a design problem is typically specified. This design problem will be available to students who later connect to the session. Next, a set of previously specified design principles are associated with the session. These are the principles that students will consider when manually critiquing designs during the session. When creating a session, instructors can specify whether anonymous users will be permitted, and if automatic critiquing will be enabled during the session.

Once a session has been created, students can connect to it using the ClassCompass client program. Students can see the design problem, but their ability to submit designs, view submitted designs, or submit critiques is determined

by the state of the session. The session state can be changed by the instructor using the session administration Web application or with the instructor version of the ClassCompass client. There are several possible session states.

—Setup. This state allows students to connect to the session and view the design problem only.
—Design. The session allows designs to be created and submitted.
—Critique. This state is used when submitted designs have been automatically exchanged among ClassCompass clients. In Critique state, students may submit critiques of the design they have been assigned.
—Open. All designs and critiques can be viewed. New designs and critiques of any submitted design may be submitted.
—Closed. All designs and critiques may be viewed but no new designs or critiques can be submitted.

These states can be used to guide a group through stages of a training exercise or to enable features for use by a collaborating development team as was described in Section 2.

3.2 Student Client

Students use a standalone Java client to interact with the ClassCompass system. This client provides support for creating and editing software designs, automatically critiquing designs as they are created, and participating in instructional sessions hosted by a ClassCompass server.

The ClassCompass client supports software design by providing a graphical editor for basic UML class diagrams. A toolbar provides access to tools for creating and placing components and connectors. Supported components include classes and interfaces. When appropriate, these components can be connected with several relationship connectors including inheritance, realization, association, aggregation, and composition. Attributes and operations can be added to classes in the diagram through a right-click context menu. This functionality for basic class diagram editing was created by extending the JHotDraw[4] graphical editor framework.

When the client is run, users are presented with a wizard for connecting to an instructional session. Users may also choose to work in offline mode without connecting to a server. In offline mode, users can create and edit design diagrams while receiving automated feedback as they work. However, offline users cannot submit or view other users' work.

When connected to an instructional session the student client supports the following additional features.

—Submit Designs. Designs may be submitted so that they can be viewed and critiqued manually by other users.
—View Designs and Critiques. Users can browse designs and critiques submitted by any participant.

[4]Available at http://jhotdraw.org.

—Critique Designs. Available designs may be critiqued. Critiques can be submitted to the session for other users to access.

3.3 Automated Critiquing

Software engineering students tend to make common mistakes when designing software. Many of these potential design flaws can be detected by analyzing a model of the UML class diagram. We have developed an automated system for detecting potential design problems based on these models and reporting them to the user. This reduces the load on expert human mentors and the immediate feedback is effective pedagogically.

This automated critiquing feature runs in the background when the user makes changes to their design. This subsystem is implemented as a collection of design critics that analyze a model of the design. Each design critic searches the design for a particular design anomaly. In most cases, the existence of such an anomaly indicates a design flaw. However, it is possible that an issue detected by a critic is not actually a problem in the context of a particular design. In either case, the user may choose not to respond to the critic.

When a critic finds a potential design problem, an entry is added to a list of critiques beside the design diagram. This unobtrusive notification allows the user to continue their work uninterrupted if they choose to ignore the automated advice. When more information about a detected problem is desired, users can click on the item of interest in the critiques box. This displays a detailed description of the critique in the Critique Details box. When applicable, the relevant portion of the design diagram structure will also be highlighted to draw the user's attention to the detected problem.

The critique detail text is typically organized into three sections.

—Critique. This describes what part of the design has been identified as a potential design flaw.

—Rationale. This section explains why the identified design element could lead to reduced software quality.

—Suggestion. When possible, this section provides suggestions on how to correct the identified design issue.

Figure 1 illustrates how an automatically detected design problem is presented to the user after more information has been requested. The critique, rationale, and suggestion components of the critique are displayed in the text area in the lower-left corner of the screen. Furthermore, the two classes (AbstractPump and DieselWaynePump) and association connector that triggered the critique have been highlighted in the diagram.

There are many critics that could be implemented to provide feedback regarding potential design problems. The problems that can be detected are those that involve structural properties of the design such as the number and types of connections between classes and interfaces. The positions and names of design elements can also be used as the basis for design critics.

ClassCompass critics are specified in Java as pluggable classes that search for a particular pattern in an object model representing the design. The object

model is a directed graph where the nodes are the classes and interfaces in the design, and the edges are the relationships between them such as inheritance and aggregation. When the user modifies the design, and thus the object model, each available critic is notified of the change. The critics then inspect the new design model to determine if the design characteristic they check for is present. For example, this may involve traversing the object model in search of a particular structural pattern. If the pattern is found, the critic will produce a critique that references the matching part of the design. In this case, a *structure selection* is also created by the critique to identify the matching structure so that it can be highlighted in the display. While this critic implementation approach prevents end-users from specifying custom critics, the specification in a general purpose language is necessary to avoid placing limitations on the kinds of critics that may be developed.

We have implemented an initial set of 19 critics in ClassCompass. The majority of these critics fall into one of three categories. Structure critics are concerned with the relationships between classes and interfaces. Naming critics identify potential sources of confusion introduced by the names of classes, interfaces, or their members. Metric critics report when the number of occurrences of some aspect of the diagram is beyond normal values. The names, brief descriptions, and suggestions provided to users by critics of each type are listed in Tables II through V.

3.4 Manual Critiquing

Manual critiquing by a human is important for identifying design problems that cannot be detected automatically. For example, the automated system is unaware of the semantics of the names in a design that suggest their function or intended usage. Many design principles require an understanding of the behavior that is suggested by a name but not formally specified in the design. For this reason, the ClassCompass system supports the manual critiquing of designs that have been submitted to the server.

Available designs are accessed and critiqued using the Critique tab of the ClassCompass client program. Designs are selected by name using a dropdown list, which causes them to be downloaded and displayed. Alternatively, a particular design may be assigned for critiquing by the system or the instructor. ClassCompass guides users by providing a list of design principles against which to evaluate the design. Selecting a design principle displays a brief description of the principle as well as session-specific instructions that provide guidance in evaluating the principle. Design principles and evaluation instructions are specified using the session administration Web application described in Section 3.1.

Critiques are specified by entering them into a text input area adjacent to the design view. Critiques are further supported with red and green structure-selection tools. These tools are activated using the toolbar and allow the user to identify parts of the design structure associated with the text critique by highlighting them in red or green. Structures are typically highlighted in green to indicate effective use of a design principle and red to indicate where design principles have been violated.

Table II. ClassCompass Structure Critics Concerned with the Relationships Between Classes and Interfaces

Structure Critics		
Critic Name	Description	Suggestion
Class References Subclass	A class references a subclass of itself	Remove the reference from the superclass to the subclass
Superclass Reference	A class references its superclass, but not through an aggregation, which could be used to make a valid Composite pattern.	Remove the association from class X to class Y or change the relationship to an aggregation
Circular Containment	There is a cycle in aggregation or composition relationships	Remove a composition or aggregation relationship between these classes
Association Cycle	There is a cycle in association relationships	Try to remove an association to break the cycle
Multiple Paths	There are two navigable paths from one class to another	If possible, remove one of the references. For example, to access the services of class X from class Z, it may be possible to make calls to class Y which can delegate the request to class X.
Duplicated Superclass Reference	A class has an association that is already defined by its superclass	Remove the duplicate association from class X to class Y
Generalizable Aggregation	A class aggregates two classes that share a superclass	Consider letting class X aggregate only instances of the common superclass, class Y, or an interface implemented by class Y
Subclass and Superclass Aggregation	A class aggregates a class and a subclass or superclass of that class	Consider letting class X aggregate only instances of class Y, or an interface implemented by Y
Unnecessary Realization	A class realizes two interfaces that extend each other	Remove the realization connection between class X and the subinterface Y

An example manual critique is shown in Figure 4. In this case, the user has downloaded another student's design and chosen to evaluate it with respect to the Liskov Substitutability Principle. The user has entered their critique and used the red structure-selection tool to identify the relevant portion of the design. This critique can now be submitted to the session to provide feedback to the designer and an example to other participants.

3.5 Instructor Client

Instructors use an instructor-specific ClassCompass client when connecting to a session. This client includes all of the student-client functionality as well as additional features for managing the session.

The instructor client allows the state of the session to be changed to any of the available states described in Section 3.1. When in critique state, instructors can initiate a design shuffle so that each connected student client

Table III. ClassCompass Naming Critics Concerned with Class, Interface, Attribute and Method Names

Naming Critics		
Critic Name	Description	Suggestion
Plural Contained Class	The target of an aggregation or composition has a plural name (which wrongly suggests that it is the container	Change the name of class X or remove the containment relationship
Method in Attribute Compartment	There are parentheses in the name of an attribute, which may occur if the user creates a method in the wrong compartment	Remove the parameters or parentheses from the attribute or delete it and define it as an operation
Get or Set Attribute Prefix	An attribute name begins with get or set, which suggests the user may have put a method name in the attribute compartment	To avoid confusion, redefine the attribute as an operation or remove the 'get' or 'set' prefix
Duplicate Class Name	Two classes or interfaces in the design have the same name	Rename one of the components

Table IV. ClassCompass Metric Critics Concerned with the Number of Occurrences of Various Design Elements

Metric Critics		
Critic Name	Description	Suggestion
Highly Coupled Design	The number of associations, compositions, and aggregations has exceeded some constant multiple of the number of classes	Reduce the number of couples in the design
Class Has Too Many Associations	A class that is not a mediator has more than some constant number of associations to other classes	Remove associations from class X to other classes, or consider decomposing the responsibilities of class X into several classes

Table V. Miscellaneous ClassCompass Critics

Miscellaneous Critics		
Critic Name	Description	Suggestion
Duplicated Members	Two classes have at least three members in common	Consider re-designing so that class X and class Y inherit their shared functionality from a common superclass
Missing Attribute	A getter and setter are defined, but no matching attribute exists	Add the attribute, newAttribute
Unnecessary Accessors	A class has more than some constant number of pairs of getter and setters, which may be an unnecessary source of clutter	Consider removing getter and setter pairs and only showing the attribute name
Redeclared Superclass Attribute	A subclass redeclares an attribute defined by its superclass.	Remove the redeclared attribute from the subclass and therefore use the one defined in the superclass, or rename the attribute in either class

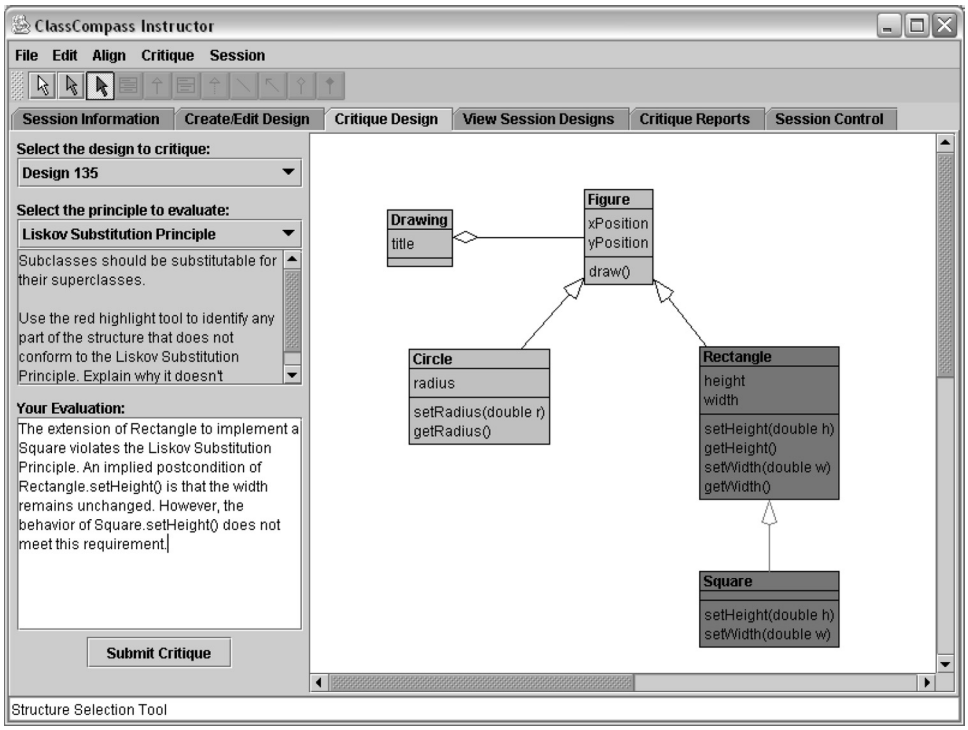

Fig. 4. The instructor-client program while submitting a critique of a design.

is assigned a design from another client to be manually critiqued. Alternatively, a single design can be selected to be sent to each student client for critiquing.

In the design or critique session states, student clients cannot be used to view designs or critiques submitted by others. This ensures that each student creates their own design without undue influence from other students' approaches. The instructor client, however, can always display any design or critique that has been submitted to the session regardless of the state. This allows the instructor to monitor the students' progress and possibly prepare for a subsequent discussion based on the designs submitted.

4. EVALUATION

The ClassCompass system has been used in several undergraduate software engineering courses at the University of British Columbia. We evaluated the system during two of the ClassCompass sessions. In the first session, students applied manual critiques to given designs. In the second session, students created designs with the aid of the automated critique system. We also provide an overview of automatic critic activations observed during several other ClassCompass sessions. The designs students created or analyzed during these sessions were small, typically involving 10 or fewer classes.

4.1 In-Class Design Principle Exercise Using Manual Critiques

The ClassCompass system was used to conduct an in-class exercise as part of a fourth-year software engineering course. The intent of the exercise was to improve students' understanding of several design principles by asking them to use the principles to manually critique a given design. Students worked in small groups with wireless-enabled laptops running the ClassCompass student client. Each group connected to a shared instructional session and opened one of two supplied designs in the ClassCompass critique view. Students then produced and submitted manual critiques of the design. The automated critiquing system was disabled during this exercise.

Approximately 70 students participated in the exercise, and 34 volunteered to complete a questionnaire. Of the 34 students who completed the questionnaire, 27 (79%) reported that they were able to apply design principles to an assigned design using the ClassCompass manual critique feature. Students also reported 10 instances in which they had difficulty understanding how to apply a design principle to the design they were critiquing. Although 12 students (35%) did not respond to the question, 13 students (38%) stated that they did not improve their understanding of design principles as a result of the tool. Six students (18%) reported that the tool did help them understand design principles. An additional 3 students (9%) reported that discussion with other students during the exercise had helped them improve their understanding of the material.

In total, 36 critiques were submitted[5]. In our opinion, 16 (44%) of these critiques clearly demonstrated an understanding of a specific design principle. An additional 8 critiques (22%) provided feedback that a designer might find helpful, although they were not related to a particular design principle.

The students were given approximately 45 minutes to learn the ClassCompass manual critiquing interface, analyze the design, and produce manual critiques. After the exercise, it became clear that it would be more effective if more time was given and several students stated this in their questionnaires. We believe that additional study is required to determine the educational value of manual critiquing when students are given adequate time to create critiques and learn from critiques submitted by other students. It would also be interesting to evaluate manual critiquing sessions where the designs critiqued are created by the participants.

4.2 Software Design Lab Using Automated Critiques

We used ClassCompass during a software design lab for students in a third-year software engineering course. These students had previously used the ClassCompass UML diagram editor during an in-class exercise, but the automatic critiquing features had been disabled. At the beginning of the design lab, students

[5]Several critiques consisting of highlighted structures without an accompanying explanation were not counted.

Table VI. Number of Activations and Resulting Improvements Observed for Each Eritic During the Software Design Lab Exercise Described in Section 4.2

Critic	Activations	Improvements
Duplicate Component Name	4	4
Redeclared Superclass Attribute	2	1
Duplicated Class Members	2	1
Plural Contained Class Name	2	1
Method in Attribute Compartment	2	1
Missing Attribute	1	1

were asked to construct a software design for a given set of requirements. Automatic critiquing was enabled, and we logged instances where design critics provided advice as well as the designs that triggered these critics.

During the one-hour lab, students worked individually or in groups of at most four students. Solutions to the posed design problem were created and submitted from 11 ClassCompass clients connected to the shared session. The logs indicate that 13 instances of automatic critiques were produced during the session[6]. Of these critiques, 9 (69%) were used by students to improve the quality of their designs. Table VI lists how often each critic was triggered as well as the number of times the student acted on each critic's advice to improve their design.

As an example of automatic critiquing, we observed that a student was notified by a critic that two classes had several members in common. The student responded by removing members from one class and making them inherit from the other class. In another case, a student created accessor methods for an attribute, but there was no corresponding attribute. After being notified of this by a critic, the student added the missing attribute. This demonstrates how the automatic critique approach can be of value to inexperienced software engineers when creating software designs.

As in the exercise described in Section 4.1, we found that the lab would be more effective if more time was given. We also found that several students did not understand why the UML editor did not allow certain operations. For example, the editor did not allow an association connector to be drawn from an interface to a class. We discuss this issue in Section 5.

4.3 Automatic Critique Sessions

The ClassCompass automatic critique features have been used during several design sessions in addition to the lab described in the previous section. During these sessions, students generally worked on given design problems or, in some cases, designs for a software engineering course project. In a typical session, approximately 10 to 15 ClassCompass clients were used for 50 minutes. During four such sessions, we observed the number of automatic critics that generated advice. Table VII lists the names of these critics and the number of times they were activated.

[6]Several additional critiques were generated that were clearly due to misinterpretation of user input. These false positives could be easily avoided with slightly more sophisticated critics.

Table VII. Automatic Critic Activations and Resulting Improvements Observed During Four Automatic Critique Sessions

Critic	Activations	Improvements
Multiple Paths	22	15
Class Has Too Many Associations	9	5
Plural Contained Class Name	8	6
Association Cycle	6	4
Generalizable Aggregation	2	2
Circular Containment	2	1
Method in Attribute Compartment	1	0
Duplicated Class Members	1	1
Subclass and Superclass Aggregation	1	1
Class References Subclass	1	0

The critics frequently activated during these sessions were predominantly structure-related. In contrast, the critics activated during the design lab described in Section 4.2 were primarily associated with naming issues. This may be due to the nature of the given design problem or the result of not providing enough time to build larger and more structurally complex designs that are more likely to have structural anomalies.

5. DISCUSSION

By observing the system during in-class exercises, we have discovered several possible enhancements that we believe would improve the tool's effectiveness.

We found that when students began creating designs with the graphical editor, they were often confused by operations that were disallowed. For example, if a student attempts to draw an inheritance connection from an interface to a class, the connection simply cannot be drawn. In this case, we believe it would be helpful if the system provided a brief explanation of why such an action is invalid. This functionality is a special case of an automatic critique that is time-sensitive; the information must be provided immediately and withdrawn after several seconds if the user has not responded to it. These critics further differ from design critics because they are activated by user action and are not a characteristic of the design model. This would allow flexibility in the activation criteria, such as only providing advice when an invalid action is repeated.

There are several ways to improve on the manual critiquing system. Students reported that they would like the ability to create and review several critiques for a design before submitting them. This would allow users to edit and possibly delete critiques as they develop their understanding of a design. Manual critiquing could be further improved if instructors are able to add comments to critiques submitted by peers or even delete them if they are entirely misleading. It may also be beneficial for students to comment on other students' critiques, but instructor comments should be distinguished from peer comments.

When there are many designs submitted in a session, it may be tedious for a single instructor to review each of them. It may be possible to develop an automatic classification and ranking scheme to assist instructors in this situation. If designs are grouped by similarity, then the instructor would be

better able to identify common errors. A ranking scheme would help instructors find and present quality solutions.

ClassCompass currently supports the creation of basic software class diagrams. When used by more advanced developers, several additional features would be beneficial. For example, the system should explicitly support multiplicities, method parameters, and visibility modifiers. Support for additional UML diagrams such as sequence and collaboration diagrams would also be useful for specifying more detailed software designs.

We have implemented an initial set of automatic critics in ClassCompass. There are many more critics that could be added to the system to improve its capabilities, such as additional metric critics (e.g., depth of inheritance) or a critic for long parameter lists. Furthermore, the system could be extended with features for disabling critics and setting their priority.

Our evaluation of the automatic critique system shows that legitimate design problems were identified and corrected as a result of critic activations. The value of the manual critique features is less apparent. To better evaluate manual critiquing, it may be useful to administer a pretest and posttest to determine whether students have improved their design skills by using ClassCompass.

6. RELATED WORK

The ClassCompass system is related to work from two different areas. The networked support for group design mentoring is related to work in the field of Computer Supported Collaborative Learning. The automatic critiquing features build on previous work in Critiquing Systems, which is a subarea of Intelligent User Interfaces.

Silicon Chalk[7] [Coatta 2002] is designed to facilitate learning in the classroom when the instructor and students have access to network-enabled computers. This software allows an instructor to broadcast various types of media to students' client applications. The system also allows the instructor to receive questions, feedback about the presentation pace, and other information during a lecture. These features are intended to support general-purpose lectures. In contrast, ClassCompass provides specific support for the instruction of software design.

The D-UML system [Boulila et al. 2003] supports distributed collaborative work on UML diagrams, enabling multiple users to interact on a shared design. The interaction is managed by policies that determine which participant has access to a given artifact at a particular time. Group collaboration is supported by awareness and communication features. Communication is supported through text messages that allow users to comment on the design. This enables a primitive form of manual critiquing, although no support for automatic critiques is provided.

The open-source ArgoUML[8] project does provide support for automatic critiques of UML diagrams. As the user works in ArgoUML, critics may can produce critiques of the diagram in a similar manner to those produced by

[7]Available at http://siliconchalk.com.
[8]Available at http://argouml.tigris.org.

ClassCompass. Generated critiques are displayed categorized by high, medium, or low priority.

Although this critiquing approach is similar to that of ClassCompass, the nature of the critiques differ. The ArgoUML critics produce critiques for a large number of tasks, many of which focus on diagram completeness. For example, after placing a class in the diagram, many critiques are produced reminding the user that the class requires a name, attributes, operations, and relationships with other classes. ClassCompass critics focus on higher-level design issues such as classes that reference subclasses or aggregations that could be pulled into a superclass. ArgoUML does not provide support for manual critiques or networked collaboration with other users.

ABCDE-Critic [Souza et al. 2000] is a tool that provides a UML class diagram editor with automatic critiquing capabilities similar to ArgoUML and ClassCompass. This system includes additional functionality that allows the user to create new critics using a logic language. ABCDE-Critic also provides a basic level of support for manual critiquing by a group of users. Group members simply use the software in sequence to add critiques in the form of annotations to the design. Unlike ClassCompass, these critiques are based only on unstructured text without guidance based on established design principles or tools for visually identifying diagram structures. Collaboration with distributed users is also not supported by ABCDE-Critic.

The Argo design environment provides critiquing functionality similar to that of ArgoUML, but the problem domain is software architecture rather than UML diagrams [Robbins and Redmiles 1998]. This system provides a graphical environment for modeling components and their relationships in a software architecture. Several categories of critics are provided such as an Evolvability critic that detects designs that have too many components at the same level of decomposition. Some of the critics can automatically correct design problems or provide a wizard to guide the user through a solution to the detected problem.

Design environments with automatic critiquing capabilities have also been developed for other domains such as kitchen floorplan layouts [Fischer et al. 1998] and medical treatment consultation [Gertner and Webber 1998]. A survey of these and other critiquing systems is provided in Robbins [1998].

7. CONCLUSION

The ClassCompass system supports both automated and collaborative support for software design education. The ClassCompass client can be used alone to create software designs; while a student works, an automated system provides assistance in the form of design critiques. Groups of students can also use ClassCompass to improve design skills by sharing designs and manually critiquing those designs with guidance based on established design principles. These automatic and manual critiquing features may also be useful for group project teams collaborating to refine a design.

We have used ClassCompass for design exercises in several undergraduate software engineering courses. Initial observations of the system have shown

that students are able to benefit from both the automatic critiques and human mentoring provided by ClassCompass. We believe the ClassCompass automatic critique system may also help to improve software product quality through better design.

REFERENCES

BOULILA, N., DUTOIT, A. H., AND BRUEGGE, B. 2003. D-meeting: An object-oriented framework for supporting distributed modelling of software. In *The International Workshop on Global Software Development, International Conference on Software Engineering*.

COATTA, T. 2002. Silicon chalk and pervasive learning using technology to support learning in many contexts. http://www.siliconchalk.com/Documentation/White-Paper-Day-in-Life.pdf.

FISCHER, G., NAKAKOJI, K., OSTWALD, J., STAHL, G., AND SUMNER, T. 1998. Embedding critics in design environments. *Knowl. Engin. Rev. 4*, 8 (Dec.), 285–307.

GERTNER, A. S. AND WEBBER, B. L. 1998. Traumatiq: Online decision support for trauma management. *IEEE Intell. Syst. 13*, 1 (Jan.-Feb.), 32–39.

LETHBRIDGE, T. 1998. A survey of the relevance of computer science and software engineering education. In *Proceedings of the IEEE Conference on Software Engineering Education and Training.*

LIEBERHERR, K., HOLLAND, I., AND RIEL, A. 1988. Object-oriented programming: An objective sense of style. In *Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'88)*, ACM Press, New York, NY, 323–334.

LISKOV, B. H. AND WING, J. M. 1994. A behavioral notion of subtyping. *ACM Trans. Program. Lang. Syst. 16*, 6, 1811–1841.

PAGE-JONES, M. AND CONSTANTINE, L. L. 1999. *Fundamentals of Object-Oriented Design in UML*. Addison-Wesley Professional Publishing.

ROBBINS, J. E. 1998. Design critiquing systems. Tech. Rep. UCI-98-41.

ROBBINS, J. E. AND REDMILES, D. F. 1998. Software architecture critics in the argo design environment. *Know.-Based Syst. 5*, 1, 47–60.

SOUZA, C. R. B., FZRREIRA JR, J. S., GONCALVES, K. M., AND WAINER, J. 2000. A group critic system for object-oriented analysis and design. In *15th IEEE International Conference on Automated Software Engineering*. 313.

Received March 2005; revised October 2005; accepted September 2006