SketchUML: The Design of a Sketch-based Tool for UML Class Diagrams

Lin Qiu
Department of Computer Science
State University of New York at Oswego
Oswego, New York 13126 USA
lqiu@cs.oswego.edu

Abstract: In Software Engineering classes, students need to learn how to use Unified Modeling Language (UML) in the software design process. In this paper, we describe a sketch-based software tool called SketchUML that allows students to create UML class diagrams as naturally as they would on paper, but with the editing capability provided by software. We describe the design choices made in SketchUML for creating a flexible paper-like interface to facilitate students to engage in learning-by-doing design exercises. Results from a formative evaluation suggest that SketchUML is capable to help students easily create UML class diagrams in class.

Introduction

Universal Modeling Language (UML) is a common language used to guide and document the software design process. UML artifacts such as use case diagrams, class diagrams, state diagrams, and interaction diagrams are created during different stages of the design process. In Software Engineering classes, students need to learn these artifacts and use them to solve software design problems. Learning-by-doing exercises can teach students these artifacts by engaging students in design activities where students need to create these artifacts to solve real problems.

In design activities, freehand sketching has been recognized as a natural and important part. (Ullman et al, 1990; Landay & Meyers, 1995; Stahovivh, 1996, Gross, 1996). Freehand sketching allows designers to quickly express and explore their design ideas in a rough visual presentation without worrying about details. For example, mechanical designers often sketch their designs on paper before moving to computer-based tools to refine their ideas (Alvarado, 2000). Research has shown that designers who sketch their designs on paper tend to create more variations of their designs than when they use drawing software. When use drawing programs, designers often refine their initial design rather than generating alternatives (Goel, 1995).

While it is important to allow students to use sketching to explore their design ideas, sketching on paper does not allow students to easily modify their designs. To edit a design, students need to cross out, erase, and redraw the part he or she wants to change. Students cannot interact with their design sketches as easily as they could with objects created in the computer (e.g., through the drag-and-drop action).

In this project, we aim to create a software tool that allows students to sketch UML class diagrams in the same way as they would on paper, but with the editing capability provided by software. We chose to support UML class diagrams because class diagrams play an important role in bridging the conceptual design of a system with the actual implementation. Our goal is to allow students to quickly engage in design exercises and focus on the design process without being distracted by the use of the software interface.

In the following, we describe the basic functionality of our software, SketchUML, and then discuss the design choices that we made to support natural sketching. We present results from a formative evaluation and discuss related work at the end.

SketchUML

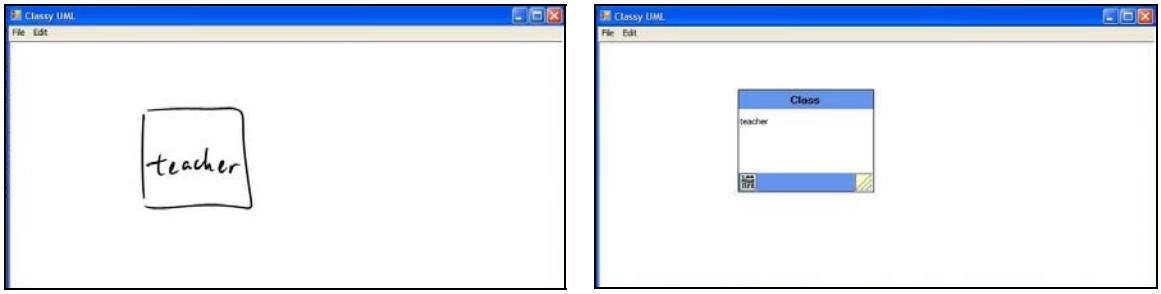


Figure 1: SketchUML transforms students' sketch into a class object.

UML class diagrams consist of class objects (the rectangles as shown in Figure 1) and associations (the links as shown in Figure 2) between these objects. SketchUML recognizes a word inside a rectangle as a class object and transforms it into a "clean" class panel (see Figure 1). A class panel has a trash bin icon on the bottom-left corner where students can click to erase the panel, and a yellow square on the bottom-right corner where students can drag to resize the panel. Students can move a panel by dragging the top blue area of a panel, in the same way as in Microsoft Windows for moving a window.

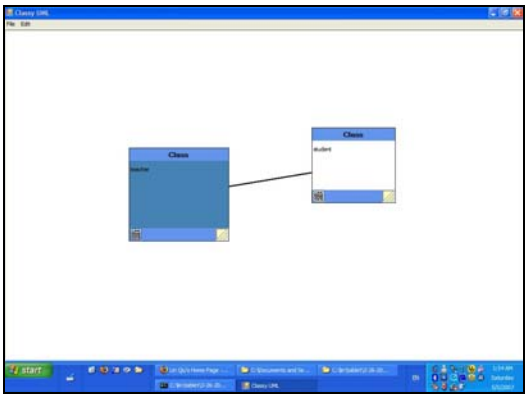


Figure 2: An association between two class objects.

Students can create an association between two classes by tapping one class object and drag a line from the selected object to the target object (see Figure 2). The association created will be a link between the centers of the two objects. When an object is moved, the associations attached with that object will move accordingly in rubber-band manner. Students can use the back tip of their pens to erase strokes, panels, or associations.

Design Choices

Our goal is to provide a paper-like interface for students to freely brainstorm ideas and practice the design of UM class diagrams without being distracted by interface issues. In the following, we discuss how we take advantage of the sketch recognition technology while avoiding its weakness in making a student-friendly sketching interface.

Sketch recognition

One critical design decision needs to be made in developing a sketch-based tool is how to balance sketching freedom and recognition accuracy. When users are given maximum freedom to sketch, it becomes very likely to misinterpret their sketches. When users are forced to sketch in a restricted way, it

becomes easier to recognize their sketches. We use a combination of geometry-based approach and graffiti-based approach to balance sketching freedom and recognition accuracy.

Research (Alvarado & Lazzereschi, 2007) and our own observation have shown that students vary in how they sketch symbols, and even individuals are not consistent during one sketching process. For example, students can draw a rectangle using four stokes (one stoke for each side of the rectangle), or one stroke that completes the rectangle from start to end. To allow students create class objects with great flexibility, we use the geometry-based approach to recognize class objects in the diagram. This approach recognizes students' sketches by the shape and relative location between strokes rather than how the strokes are drawn. It allows students to draw a rectangle in any number of stokes by any order. Students do not need to follow a pre-defined manner to create class objects. They can create a class object by sketching a rectangle and then writing the class name inside, or writing the class name first and then drawing a rectangle around it. We further improve recognition accuracy by only considering a rectangle with words inside as a class object. This avoids the mistake of incorrectly recognizing a rectangle (e.g., recognizing a big letter "O" as a rectangle) and turning it into a class object.

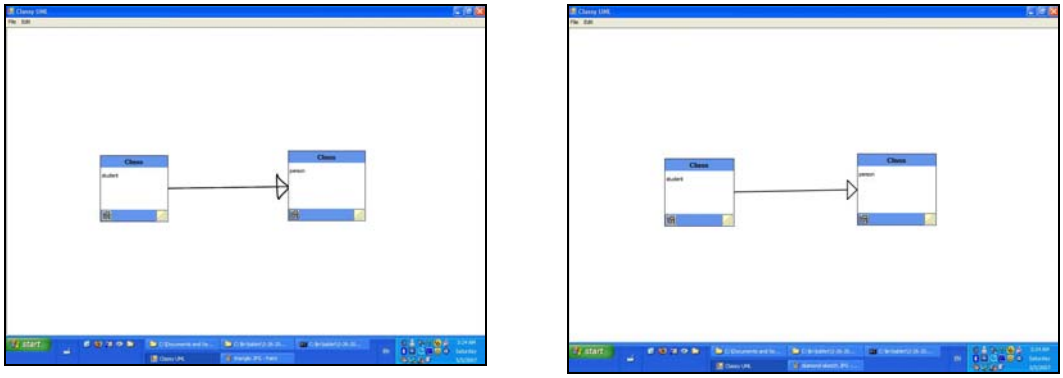


Figure 3: Annotating a triangle on an association turns an association into a generalization.

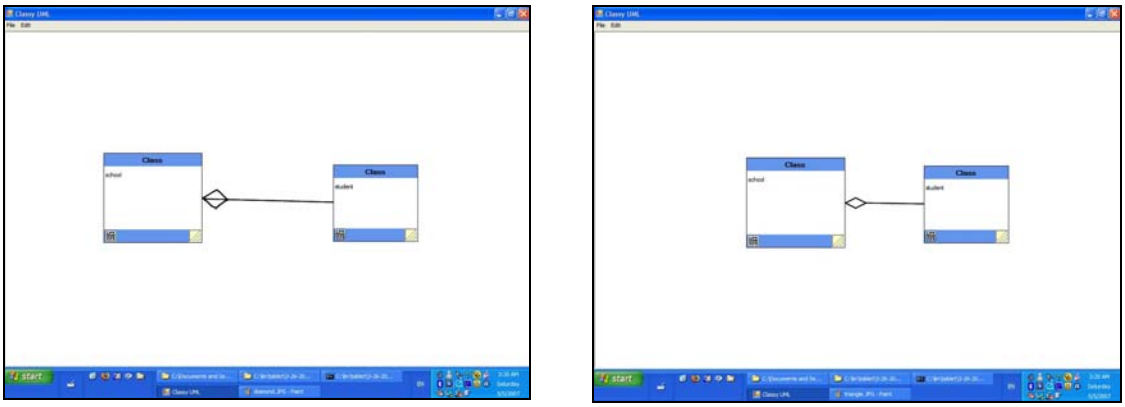


Figure 4: Annotating a diamond on an association turns an association into an aggregation.

We use a graffiti-based recognition approach to recognize two gestures, triangle and diamond, for annotating an association as a generalization association or an aggregation association respectively. Figure 3 shows that students add a triangle onto an association and SketchUML turns the association into a generalization. Figure 4 shows that students add a diamond onto an association and SketchUML turns the association into an aggregation. The graffiti-based approach requires students to draw the triangle and diamond in a single stoke that starts and ends at the same point. While this puts constraints on the way how students draw these two gestures, we consider it reasonable because these two gestures are fairly easy to complete in a single stroke. Furthermore, these gestures are standard annotations in UML diagrams. It should be natural for students to use them to annotate associations.

We believe the use of geometry-based approach for commonly used symbols and graffiti-based approach for few occasionally used gestures balances sketching flexibility and recognition accuracy. This strategy provides great flexibility for sketching the majority of the design while accurately recognizing the gestures for annotating the design.

Sketch transformation

Recognition feedback can be provided to the user by either giving the feedback immediately when objects are recognized (e.g., immediately transforming a class object when the user is sketching), or waiting until the user finishes the whole design, or letting the user tell the system when to transform, (e.g., by clicking a "show" button).

Previous work (Alvarado, 2004) suggested that users could be distracted by recognition feedback during their design process. It is acceptable to ask the users to decide when the system should perform sketch transformation. However, this requires users to leave their design task and perform an interface operation to trigger the transformation. Furthermore, if the user chooses to perform transformation after he/she finishes sketching, the user will have to verify the transformation of each component in the design. For example, if there are 20 class objects and 10 associations, users have to go through them one by one and verify them. If there is any recognition error, users have to modify their original sketches or the transformed diagram to correct them. The verification process could be particularly difficult if the user has already left his/her design context and needs to go back to think about the design rationale again. Furthermore, Hammond and Davis (2002) reported that users preferred working with "clean-up" objects than their original sketches. Therefore, we chose to automatically transform users' sketches as soon as they are recognized by the system.

To automatically transform users' sketches while the user is still sketching, it is important to avoid interfering users' design process. We chose to perform sketch transformation when the user's pen tip is no longer detected by the sketching surface (the user has moved his/her pen away from the screen), or no ink has been drawn for 5 seconds. When either of the above two conditions occurs, we assume the user is no longer in the middle of constructing an object. We believe this just-in-time transformation approach allows users to verify the recognition results in context. It would be natural for users to correct recognition errors when they are still working with the object. In addition, after the transformation, users can easily perform editing operations (such as drag-and-drop and resizing) on the objects. This would not be possible if the user still works with their original sketches.

Implementation

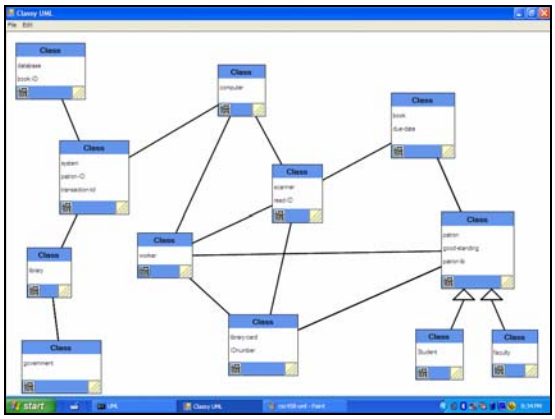
SketchUML is implemented in C# on the Microsoft .NET 3.0 framework. We use Microsoft Tablet PC SDK for ink collection and the ink analysis package for ink and gesture recognition. The ink analysis package supports geometry-based recognition by providing a tree of recognized primitive objects such as words and shapes in users' sketches. The package also supports the recognition of a dozen predefined gestures using the graffiti-based approach.

SketchUML is currently been developed and used on HP Tablet PCs running Windows XP. UML class diagrams created in SketchUML are saved in XML files.

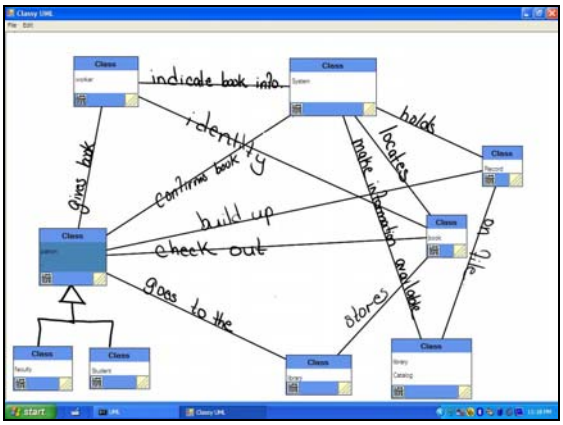
Formative Evaluation

We conducted a formative study of SketchUML in a software engineering class with four undergraduate students (see Figure 5). All students had used a CASE (Computer Automated Software Engineering) tools of their own choices (e.g., Visio or JTogether) in a homework exercise before the evaluation study. During the study, the author gave a short demo of how to sketch objects and create associations in SketchUML. Each student was given a Tablet PC to create a class diagram for a school library system according to a given "check out books" use case. The author observed how students worked with SketchUML during the study. After students finished their diagrams, they completed a survey regarding their experience with SketchUML.

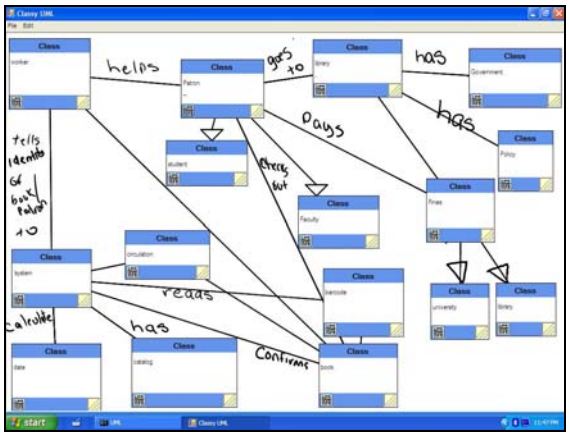
Figure 5: Students using SketchUML.



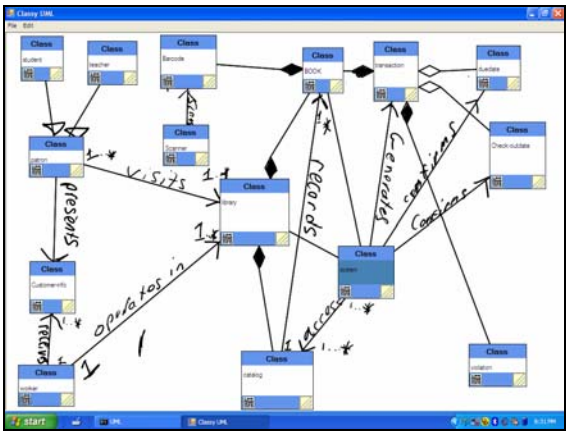
1



2



3



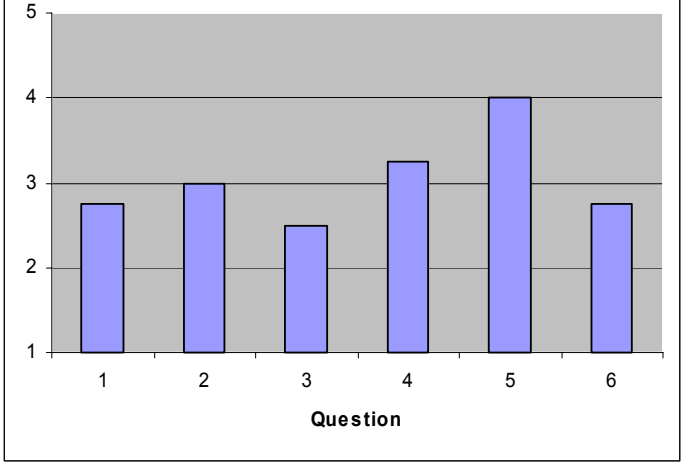
4

Figure 6: UML class diagrams created by students during the evaluation study.

Figure 6 shows the class diagrams created by the students. The diagrams along with our observation revealed two interesting findings.

First, we found that students took advantage of sketching flexibility to create objects that are not yet supported by the system. For example, students were told that SketchUML currently did not support association names and multiplicity. However, three of the four students still wrote names on associations. One student even filled ink into diamonds to distinguish solid diamonds (for compositions) from empty diamonds (for aggregations) (see the 4th diagram in Figure 6). This would not be possible if students had used a tool that required them to create objects by selecting an item from a toolbar first and then create it on a canvas. The flexibility of the sketching interface allows students to create objects beyond what the system can support. This is particularly important because we want students to be able to freely express their design. We do not want the limitation of the software to confine students' design capability.

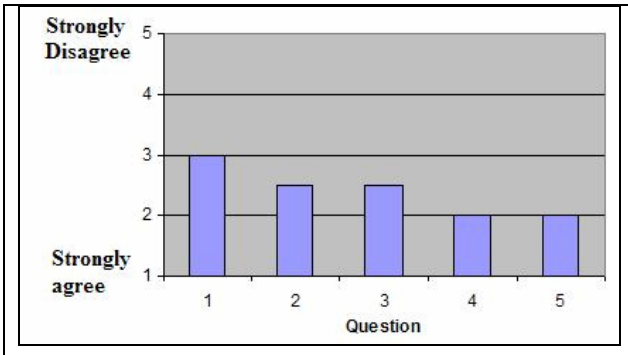
Second, we found that students adapted their sketching style to the capability of the software. In diagrams shown in Figure 6, all class objects are "clean" objects created by the system. There is no class object consists of students' original ink. In contrast, in diagram 2, 3, and 4, there are diamonds and triangles that remain in students' original ink. We believe that this is caused by the difference between the recognition accuracy for class objects and the recognition accuracy for diamonds and triangles gestures. Currently, the recognition accuracy for class objects is much higher than the recognition accuracy for diamonds and triangles gestures. When students realized that the system can relatively accurately recognize their class objects, they were willing to spend effort to correct recognition errors to have all their class objects correctly recognized by the system. When students realized that the recognition accuracy for diamonds and triangles gestures was relatively low, they gave up trying to make "perfect" triangles or diamonds to make the system recognize their drawings. This suggests that there is a recognition accuracy threshold. When the recognition accuracy is higher than the threshold, students will try to fix recognition errors and have their sketches recognized by the system. When the recognition accuracy is lower than the threshold, students will not try to draw their sketched "perfectly" to satisfy the system, and will leave their sketches in original strokes. Future research is needed to understand how many recognition errors students can tolerate and how high the recognition accuracy need to be to keep students from giving up on the system.



1. How difficult is it for you create a class diagram?
2. How difficult is it for you to create a class object?
3. How difficult is it for you to create an association between class objects?
4. How difficult is it for you to create an aggregation association?
5. How difficult is it for you to create a generalization association?
6. Overall, how difficult is it for you to use the system?

Figure 7: Students' responses to questions regarding the usability of SketchUML.

Figure 7 shows students' responses to questions regarding the usability of SketchUML. On a scale from one to five, with 1 being "very easy" and 5 being "extremely difficult", students reported that overall it was not difficult to use SketchUML. However, students felt difficult to create generalizations and aggregations. This is due to the low recognition accuracy for the diamond and triangle gestures. We are working on this problem by using more contextual information to improve the recognition accuracy.



1. I prefer to use the system to construct a UML class diagram over sketching it on paper.
2. I prefer this system over other systems that I have used for constructing UML class diagrams.
3. Overall, the system is easy to use for constructing a UML class diagram.
4. Overall, it is pleasant to use the system.
5. I would recommend this system to be used in other classes.

Figure 8: Students' responses to questions regarding their experience in the evaluation study.

Answers to Question 2 shown in Figure 8 suggest that students preferred SketchUML over other CASE software tools. This is consistent with the answers that students gave when they were asked "How do you compare this system to other systems that you have used for constructing UML class diagrams?" Students responded with 2.25 on average on a scale from one to five with 1 being "much better" and 5 being "much worse."

While students preferred SketchUML over other CASE tools, answers to Question 1 shown in Figure 8 suggests that students did not prefer SketchUML over paper-sketching. This is largely due to the problems that students mentioned in their answers to "What do you like least about the system:"

"The fact that it does not have a undo function and the eraser tool is not so specific. It would erase everything in that particular path instead of what you indicated it to do."

"Aggression and generalization are hard to draw on the screen. Also it is very easy to make a mistake that can cause you to lose your work. Maybe an undo button /save option."

"The system should allow for association to be recognized. Also, the system should recognize multiplicity."

The first problem is that the system could not accurately recognize the diamond and triangle gestures for creating aggregations and generalizations. The second problem is erasing. SketchUML allows users to use the back tip of the digital pen to remove an object or an association in the same way as in paper-based sketching. However, objects in the software cannot be erased little by little. An object is completely erased when the back tip of the pen performs a scratching action on it. In contrast, an object is erased gradually in paper-based sketching. The rubber only erases the part that it touches on paper. The whole object does not disappear all in a sudden. This allows the user to quickly realize unintended erase before the whole object disappears and reconstruct the part that is accidentally erased. The lack of incremental feedback in erasing in sketching tools makes it difficult for users to stop the erasing action in time. We are addressing the above problems in our current work.

When students responded to "What do you like most about this system," they wrote:

"The system is more intuitive than other programs that do not allow drawing and only use toolbars + symbols to construct diagrams."

"The text recognition worked well overall except for aggregations. Associations were very intuitive (tap once on a class and draw a line). Dragging around classes also worked well. "

"I like how the user can just draw whatever they want on the screen. With other software systems, I had to read the manual to learn about the different functions. With this system, you can just draw it on the screen."

"The system allow for faster creation of class diagrams. I think this is a very big advantage over other software."

These comments suggest that our software provides a flexible and intuitive user interface for fast creation of class diagrams.

Related Work

Tahuti (Hammond & Davis, 2002) is a sketch-based tool for UML class diagrams. It uses a multi-layer recognition algorithm to recognize sketches by their geometrical shapes and therefore gives users the freedom to sketch a object in different ways. Tahuti displays user sketches and the recognition results in two different views. The draw view provides a canvas for sketching, and the interpreted view displays the recognition results. Users can switch between the two views and perform editing such as drag-and-drop in either view. SketchUML differs from Tahuti in that it immediately replaces recognized sketches into objects. Users do not need to switch to a different view to see the recognition results. Tahuti currently does not support text recognition. It requires users to enter texts through a virtual keyboard. Our system supports text recognition so that users can directly write texts inside class objects.

TabletUML (The Tablet UML Company, 2007) and Ideogramic (Damm, Hansen, & Thomsen, 2000) are two commercial sketch-based tools for UML. While they fully support UML artifacts such as class diagrams, use case diagrams, sequence diagrams, and state diagrams, they require users to use a set of predefined gestures to create objects. To use them in class, students have to memorize the gestures before engaging in design activities. In contrast, SketchUML only requires students to use two predefined gestures, diamond for aggregation and triangle for generalization. Besides these two gestures, students can sketch as naturally as they would on paper.

Sketch-based tools have been built in other domains such as mathematics (LaViola, 2005), electrical engineering (Gennari, Kara, & Stahovich, 2005), and chemistry (Tenneson, 2005). ASSIST (Alvarado, 2000) is a tool that allows students to sketch mechanical systems and see a simulation of their design in a two-dimensional kinematical simulator. Students can sketch mechanical objects such as springs, blocks, and wheels, and observe the behavior of these objects under the influence of gravity and physics laws. SILK (Landay & Myers, 2001) is a tool for user interface design. It allows users to sketch a prototype of a user interface and recognizes the interface widgets contained in the sketches. It can generate a storyboard based on the links between the interface screens specified by the user. For example, the user can specify that a click on a button on screen A leads to screen B by linking the button on screen A with screen B. In the simulation mode, SILK will let users click on button A and move on to screen B.

CASE tools such as Together (Borland, 2006) can be used to create UML diagrams on the computer. While they provide powerful editing features, they often have complex menu items and toolbars that are too complicated for students to quickly learn how to use them. Students have to spend extra effort familiarizing with the tools before engaging in learning-by-doing exercises. Furthermore, these tools often require the user to click a button to tell the system which component he or she will draw and then draw the component on the canvas. The use of interface components makes it inflexible to create the design and can easily distract users from their design process.

Conclusion

In this paper, we described the design of a software tool called SketchUML to allow students to sketch UML class diagrams naturally as they would on paper but with editing capabilities on a computer. SketchUML recognizes students' sketches and turn them into classes and associations correspondingly. It balances sketching flexibility and recognition accuracy by using a geometry-based approach to recognize common elements in the diagram and using a graffiti-based approach to recognize special gestures. It provides immediate recognition feedback to allow students to correct recognition results in context. Formative evaluation results show that students adapted their sketching style to the capability of the software and their sketching went beyond what the systems could support.

References

Alvarado, C (2000). A natural sketching environment: Bringing the computer into early stages of mechanical design. *Master's thesis*, Massachusetts Institute of Technology, 2000.

Alvarado, C. (2004) Sketch Recognition User Interfaces: Guidelines for Design and Development. In *Proceedings of AAAI Fall Symposium on Intelligent Pen-based Interfaces*, 2004.

Alvarado, C., and Lazzereschi, M. (2007) Properties of Real World Digital Logic Diagrams. Submitted to *1st International Workshop on Pen-based Learning Technologies*.

Borland. (2006) http://www.borland.com/products/downloads/download_together.html

Damm, C. H., Hansen, K. M., & Thomsen, M., (2000). Tool support for cooperative object-oriented design: Gesture-based modeling on an electronic whiteboard. In *CHI 2000*. CHI.

Gennari, L, Kara, L. B., & Stahovich, T. F. (2005) Combining geometry and domain knowledge to interpret hand-drawn diagrams. *Computers and Graphics: Special Issue on Pen-Based User Interfaces*, 2005.

Goel., V. (1995). *Sketches of Thought*. MIT Press. Cambridge, Mass, 1995.

Gross, M. D. (1996). The electronic cocktail napkin - a computational environment for working with design diagrams. *Design Studies*, 17:53¨C69.

Hammond, T., & Davis, R. (2002) Tahuti: A Geometrical Sketch Recognition System for UML Class Diagrams. In *Proceedings of 2002 AAAI Spring Symposium on Sketch Understanding,* 2002.

Landay, J. A., & Myers, B. A. (2001) Sketching Interfaces: Toward More Human Interface Design." *IEEE Computer*, vol. 34, no. 3, March 2001, pp. 56-64.

LaViola, J. (2005) Mathematical Sketching: A New Approach to Creating and Exploring Dynamic Illustrations, *Ph.D. Dissertation*, Brown University, Department of Computer Science, May 2005.

Stahovich, T. (1996). Sketchit: a sketch interpretation tool for conceptual mechanism design. *Technical report*, MIT AI Laboratory.

Tenneson, D. (2005) Technical report on the design and algorithms of chempad. *Technical report*, Brown University, 2005.

The Tablet UML Company. (2007) http://www.tabletuml.com/.

Ullman, D. G., Wood, S., and Craig, D. (1990). The importance of drawing in mechanical design process. *Computer & Graphics*, 14(2):263¨C274.

Acknowledgements

This work was supported primarily by the HP Higher Education Technology for Teaching Grant 2006. We would like to thank Jesse Arens and Richard Buck for helping to implement the software.