

SourceSight Code Navigation System

Project Phase III

Submitted By:

Lior Berry	(89457030)
Wesley Coelho	(55013007)
Ed McCormick	(89251037)
David Sprague	(88023031)
Trevor Young	(89252035)

Instructor: Joanna McGrenere
CPSC 544: Human Computer Interaction
Tuesday, December 16th, 2003

Section 1: Re-Design Rationale	3
SourceSight Redesign	4
Cross-Package Relationship Visualization and Navigation.....	4
Enhanced landmarks	4
Shortcut keys.....	4
Method-level relationship view	5
Automatic diagram layout.....	5
SourceSight High-fidelity Vertical Prototype.....	6
Main Window	6
Package Layout View	8
Class View	9
Tool-tips and Method Specification View	10
Method Edit Window.....	11
Hierarchical Tree View.....	12
Radar View	12
Section 2: User Evaluation	13
Evaluation Protocol.....	14
Methods.....	14
Subjects.....	14
Materials	14
Consent	15
Questionnaires.....	15
Other Forms	16
Procedure	16
Results.....	17
Questionnaire 1	19
Questionnaire 2	20
Discussion.....	20
Confounds.....	21
Guidelines for Future Studies	22
Final Design Rationale and Discussion	23
State of the Design.....	23
Positive Usability Aspects of SourceSight	23
Negative Usability Aspects of SourceSight.....	24
Predicted Usability within Industry	25
Section 3: Appendices	26
Appendix A: Interview Questionnaires	27
Section 1: Preliminary User Experiences	27
Section 2: Comparing Eclipse and SourceSight	28
Appendix B: Data Analysis	30
Univariate Analysis of Variance (Task A).....	30
Univariate Analysis of Variance (Task B).....	31
Estimated Marginal Means	33
Appendix C: Reference Sheets	34
Eclipse (Modified) Quick Reference	34

SourceSight Quick Reference	36
SourceSight UML Reference	38
Appendix D: User Introduction	40
Read Out Instructions	40
Appendix E: SourceSight Bug List.....	44
Appendix F: External Media.....	45
Appendix G: Evaluation Tasks	46
Task A.....	46
Task A Solution	46
Task B	48
Task B Solution.....	48
Appendix H: Sample Consent Forms	50
Sample Form One	50
Sample Form Two.....	53
Sample Form Three.....	56
Appendix I: Completed Questionnaires.....	58

Section 1: Re-Design Rationale

SourceSight Redesign

The Phase II task-oriented walkthrough and informal user evaluation results identified several shortcomings in the SourceSight design. Consequently, we re-visited aspects of our low-fidelity prototype to alter and enhance some of our previous design decisions to better support user tasks. Though important, most of the changes identified during this redesign exercise were not part of the core features implemented in the Phase III vertical prototype. Each of the items that were modified during this iteration is discussed in detail below.

Cross-Package Relationship Visualization and Navigation

It is often the case that dependencies exist between classes that lie within different packages. However, when working with the Phase II low fidelity prototype it became clear that navigation from one package to another would be somewhat cumbersome. In constructing a dependency, it was necessary to navigate to the other package by moving ‘up’ to the parent package diagram and then drilling down into the other package to see the other class. Furthermore, the relationships between classes in different packages were not visible. To correct this problem, we introduced a menu option that allows a user to toggle a view that displays cross-package relationships. When cross-package relationships are displayed for a particular class *MyClass*, a package is displayed on the diagram if there is a relationship between one of its classes and class *MyClass*. An association line connects the package to class *MyClass*. Double-clicking on a package navigates to the class view for that package and the original class *MyClass* now appears on this diagram. However, class *MyClass* appears somewhat faded to indicate that it belongs to another package.

Enhanced landmarks

The previous iteration included a feature for saving landmarks. Landmarks are essentially x-y coordinate bookmarks on a diagram. Landmarks are useful for remembering a location in a diagram where work is often carried out. However, often the set of classes that are frequently accessed are not ‘geographically’ centrally located on a diagram. This iteration’s enhanced landmark feature allows a user to create special diagrams and add classes to them from anywhere in the system. The classes in these diagrams function as pointers to the classes in the real system. These enhanced landmarks allow user-defined collections of classes to be easily accessed from one location. However, any relation between the classes is visible, thus providing key information that would be absent from a simple list of classes.

Shortcut keys

Creating graphical representations is usually more time consuming than creating equivalent text-based representations. Creating an empty class diagram in SourceSight, and therefore a program skeleton, requires repetitive movements between the diagram and the toolbar. The appropriate button on the toolbar must be clicked to create a program element such as a class, and then the canvas must be clicked to indicate where the class is

to be placed. Placing another class requires another trip to the toolbar and back to the canvas. To improve efficiency, this iteration of SourceSight includes shortcut keys to activate the creation tools in the toolbar. For example, ctrl + alt + 'c' will select the class creation tool. This can be used repeatedly to quickly place several class figures on the diagram.

Method-level relationship view

The SourceSight class view provides a somewhat high-level overview of the interactions between classes. During the informal evaluations, users expressed a desire to see such interactions for particular methods. Therefore this iteration includes a right-click popup menu option for methods that displays line connections (relationships) between that method and methods in other classes. The line connections in this view are generated automatically from the source code.

Automatic diagram layout

Several users in Phase II indicated that they were concerned about time that might be spent maintaining the diagram. Although the diagram and source code are part of the same development artifact in SourceSight, the diagram layout does not affect how the program executes and it is not necessary to maintain a sensible layout to use SourceSight. Therefore, users will need to spend some time arranging the diagram to make it easier to understand. To reduce diagram maintenance time, we decided that SourceSight should include a feature that automatically lays out elements of the diagram if the user does not want to manually perform this task. The auto-layout feature attempts to arrange items with minimal edge crossing and obeying UML conventions. An example UML layout convention is that super classes are always placed above subclasses.

The Phase III vertical prototype includes a feature that enables source code files to be imported into SourceSight with a basic initial layout. This is crucial for converting existing projects into SourceSight and was also required for the testing.

SourceSight High-fidelity Vertical Prototype

We created a high-fidelity prototype in order to conduct a user evaluation of SourceSight. The prototype implements a vertical set of features that support core programming tasks. Although the implemented features are a small subset of the SourceSight system, we believe that they are sufficient to evaluate the novel navigation and visualization approach.

Main Window

The main window of the prototype contains the following components:

- **Menu Bar.** The menu bar contains menu items for opening and saving program code files, toggling the package explorer view, and several other features for manipulating graphical figures in a program diagram.
- **Tool Bar.** The tool bar provides access to tools for creating components of a program, including Java packages, classes and interfaces. The tool bar also contains tools for creating connections between components that symbolize relationships such as inheritance and aggregation. Finally, the last button on the toolbar allows the user to navigate to the parent diagram of the currently displayed diagram.
- **Diagram View.** The diagram view dominates SourceSight's main window. This is the large area in the center where diagrams are displayed for the user to interact with.
- **Radar View.** The radar view is displayed in the lower left corner of the main window. The Radar View is described in the "Radar View" section.
- **Tabbed Multi-View Panel.** This is located on the bottom of the main window adjacent to the Radar View. This panel is used to display a number of views that provide a variety of information to the user. For example, the Specification window/tab displays the documentation for the diagram element that the mouse is currently hovering over in the diagram view. Other tabs in the Multi-View Panel are not yet implemented.

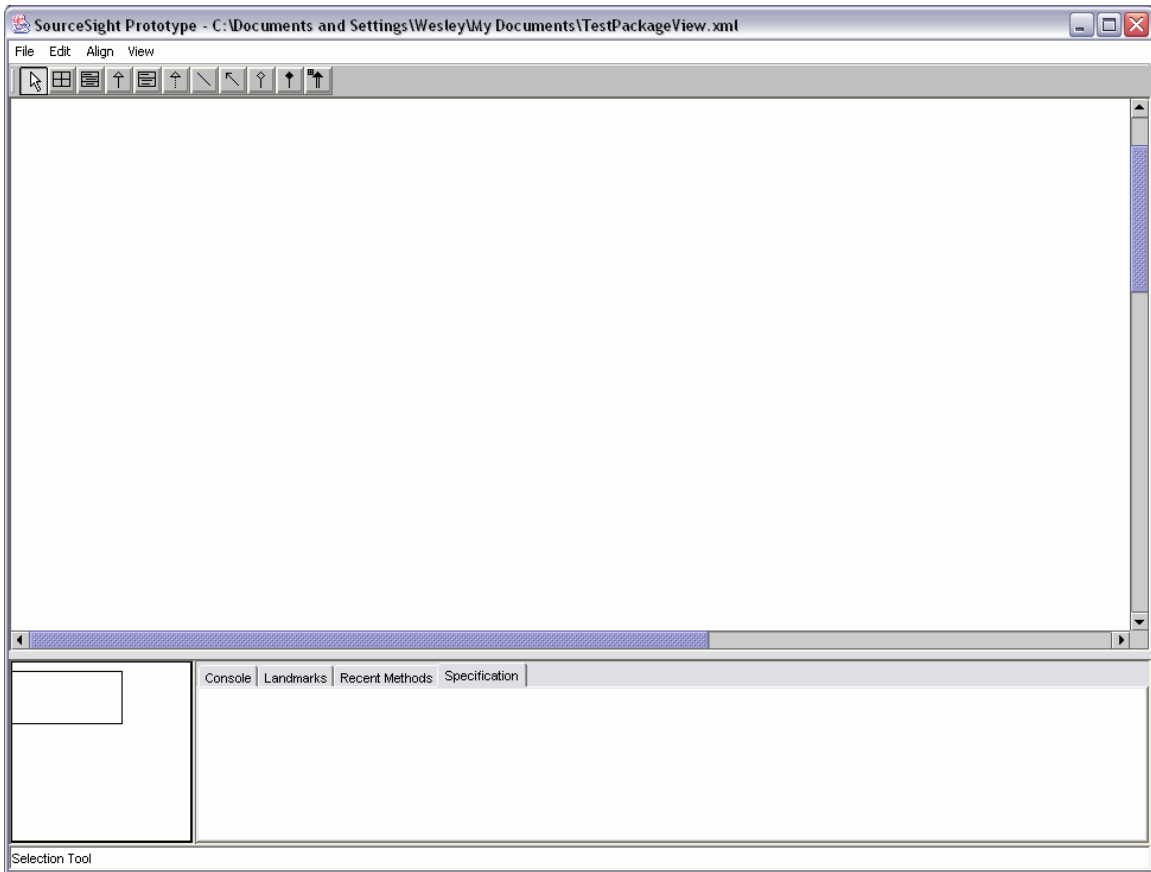


Figure 1. SourceSight Main Window

Package Layout View

When a project is first opened in SourceSight, the user is typically presented with a graphical layout of the Java packages that contain the source code for the program. Figure 2 illustrates several packages displayed in the diagram view of the main SourceSight window. Graphical figures that represent packages display the <<Package>> stereotype. Association arrows between package figures indicate that a package depends on another package. Hovering over a package figure will display a tool-tip containing the specification or descriptive documentation for that package. Double-clicking a package will drill down into the package by displaying a diagram of the classes in that package. Users can then press the last button on the toolbar to return to this view.

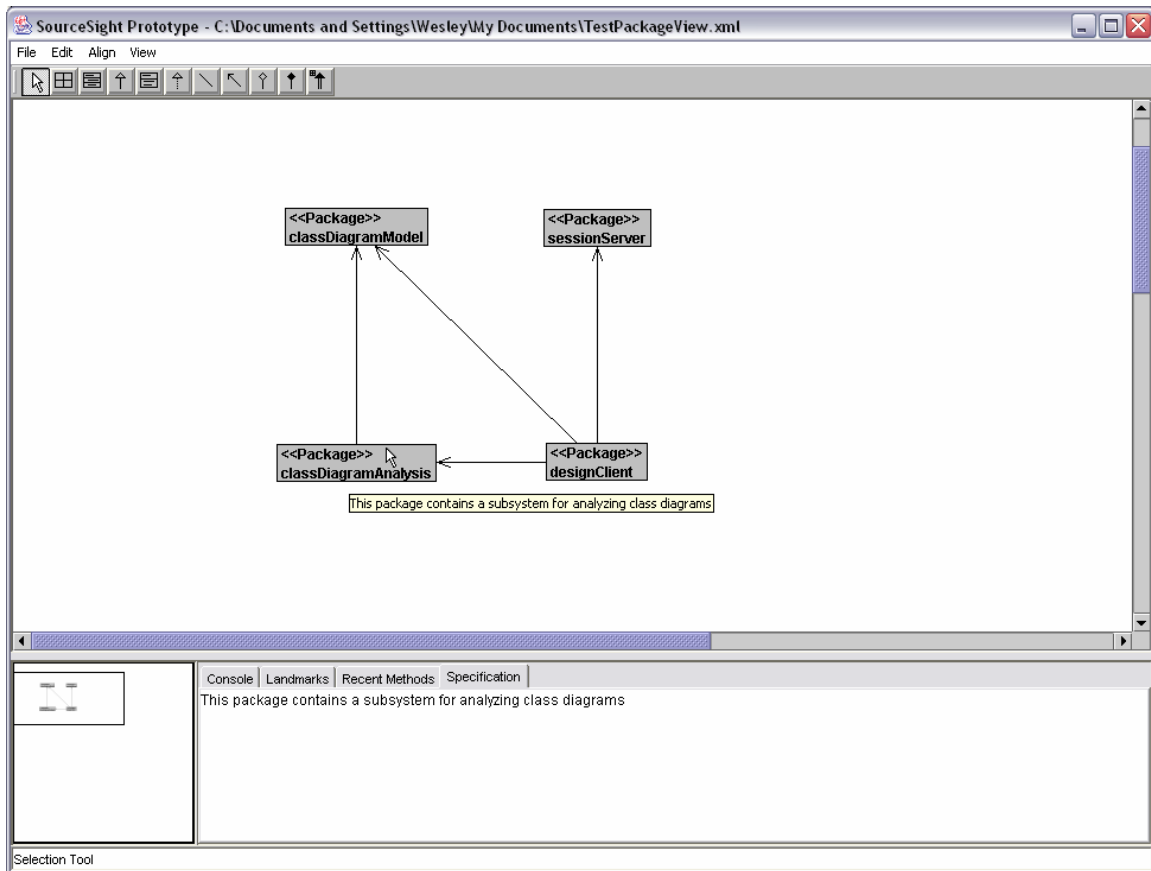


Figure 2. Package Layout View

Class View

After drilling down into a package, a diagram of the classes in that package is displayed. This diagram also displays Java interfaces and the relationships (arrow connectors) between class and interface figures. Users can navigate this view using the scrollbars or the Radar View.

This view can be used to add components to the program being edited. To add a class, interface or cross-package relationship users can select the appropriate creation tool from the toolbar and click the diagram where the component is to be placed. Right-clicking on a class or interface figure will allow users to add fields or methods via a pop-up menu. Users can add connection relationships by using the appropriate tool from the toolbar.

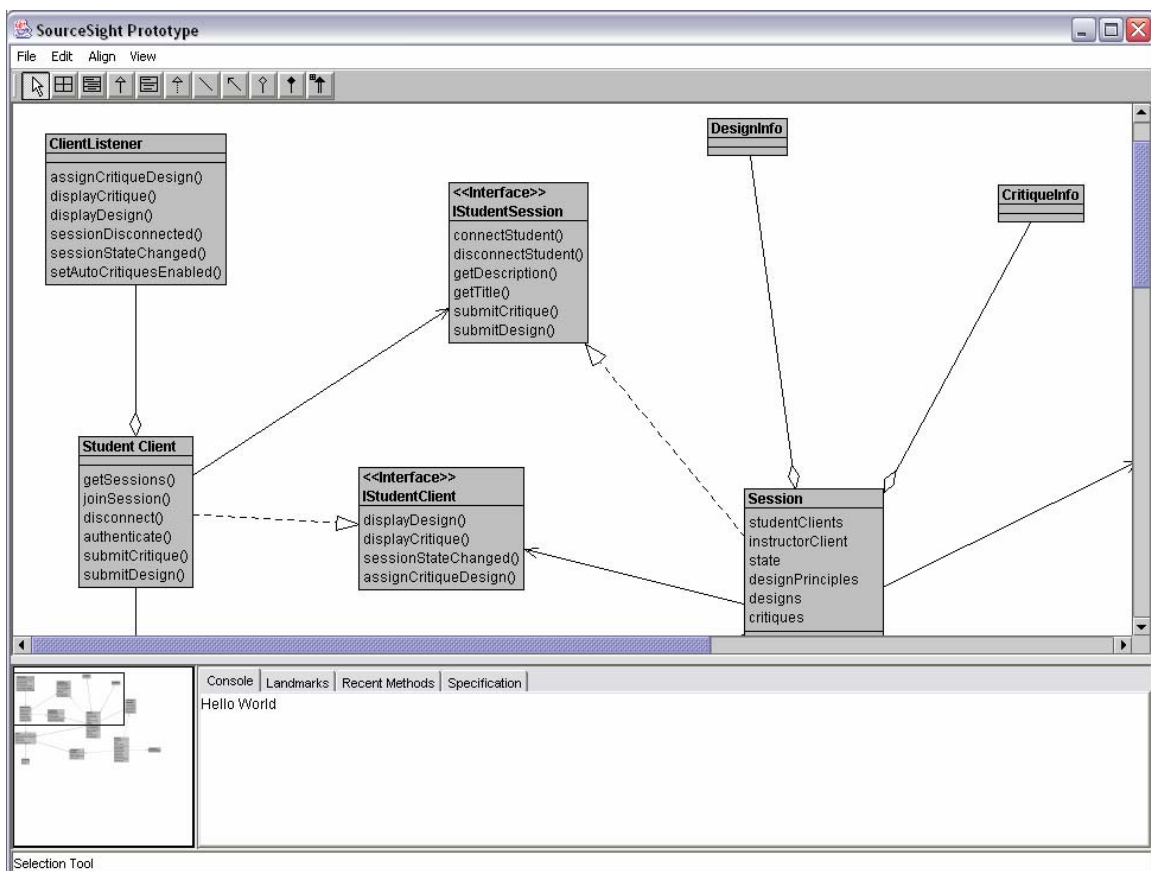


Figure 3. Class View

Tool-tips and Method Specification View

To allow users to quickly and accurately determine the purpose of a method without needing to read its source-code implementation, the Class View provides tool-tips that display a method's specification simply by hovering over it with the mouse. Similarly, tool-tips are displayed when the mouse hovers over fields or the name of a class. In Figure 4 the user is hovering over the name of a class to get more information about its function. If the user needs a longer look at the documentation than the transient tool-tip provides, the "Specification" tab can be selected in the Multi-View Panel to view the specification for the last method hovered over.

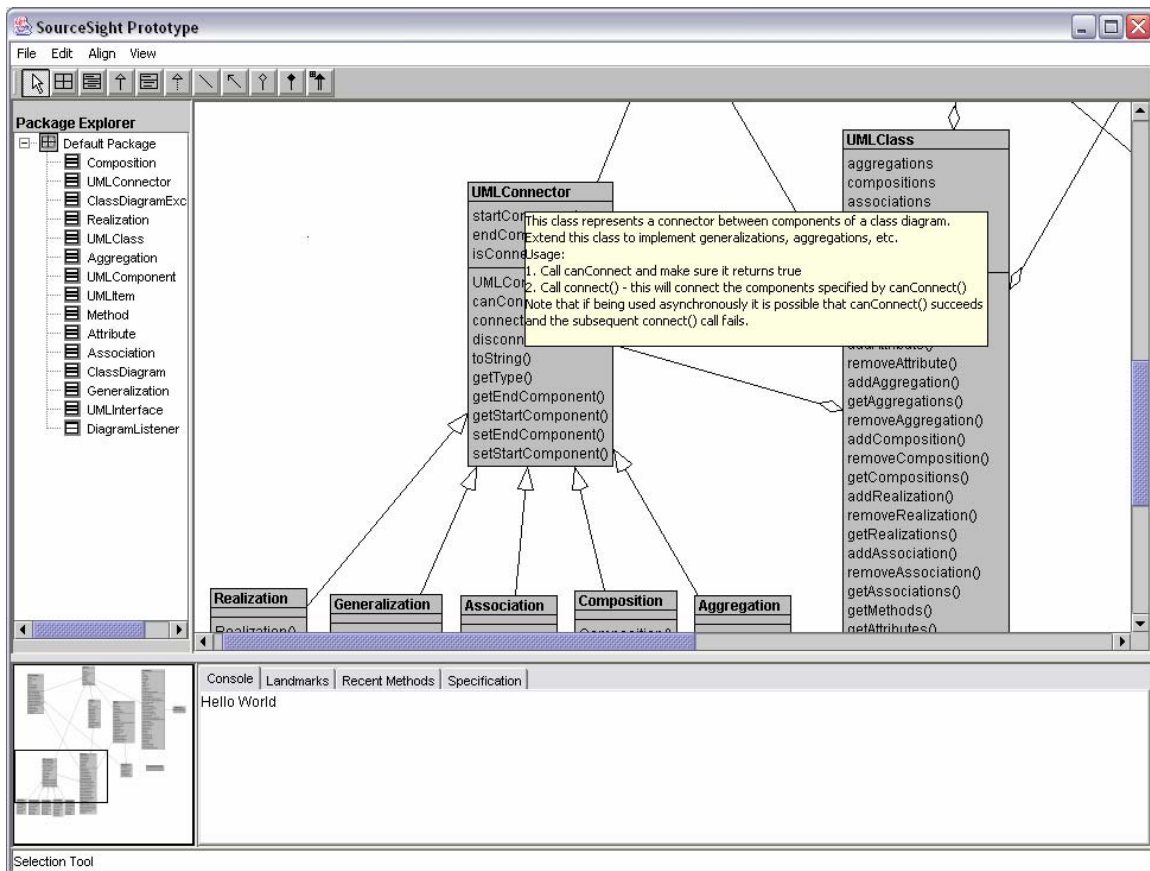


Figure 4. Class View with tool-tip specification

Method Edit Window

The user will often need to see the actual implementation of a method or edit its specification. This will typically occur when the user is creating a new method or if the tool-tip information suggests to the user that they will need to further investigate a particular method. To view a method's implementation, users can double-click a method in the diagram to open a Method Edit Window. The Method Edit Window uses a movable divider to separate it into two parts: one part for the specification text and another for the program code. Users can simply edit the code or specification text in this window to modify it. A Method Edit Window is shown in Figure 5.

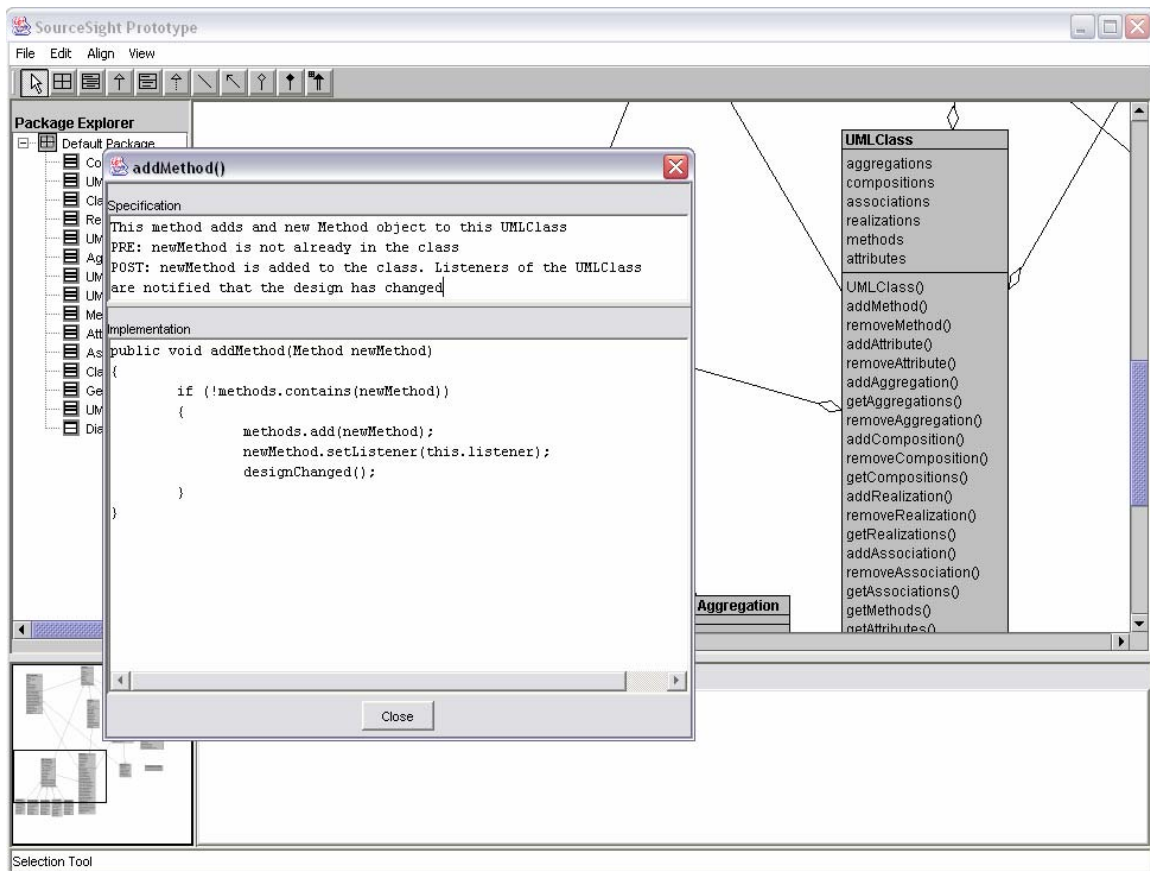


Figure 5. Method Edit Window

Hierarchical Tree View

A hierarchical tree view panel may optionally be turned on to assist with navigation. This view, called the Package Explorer, is displayed on the left side of the screen in Figure 6. In this prototype this view essentially provides a list of classes and interfaces currently present in the main diagram view. Clicking on a class in this view does not cause the code for the class to be displayed, as with a traditional IDE tree view. Instead, the corresponding class's figure is centered in the main diagram view so that the user can see the context of its interaction with other program elements.

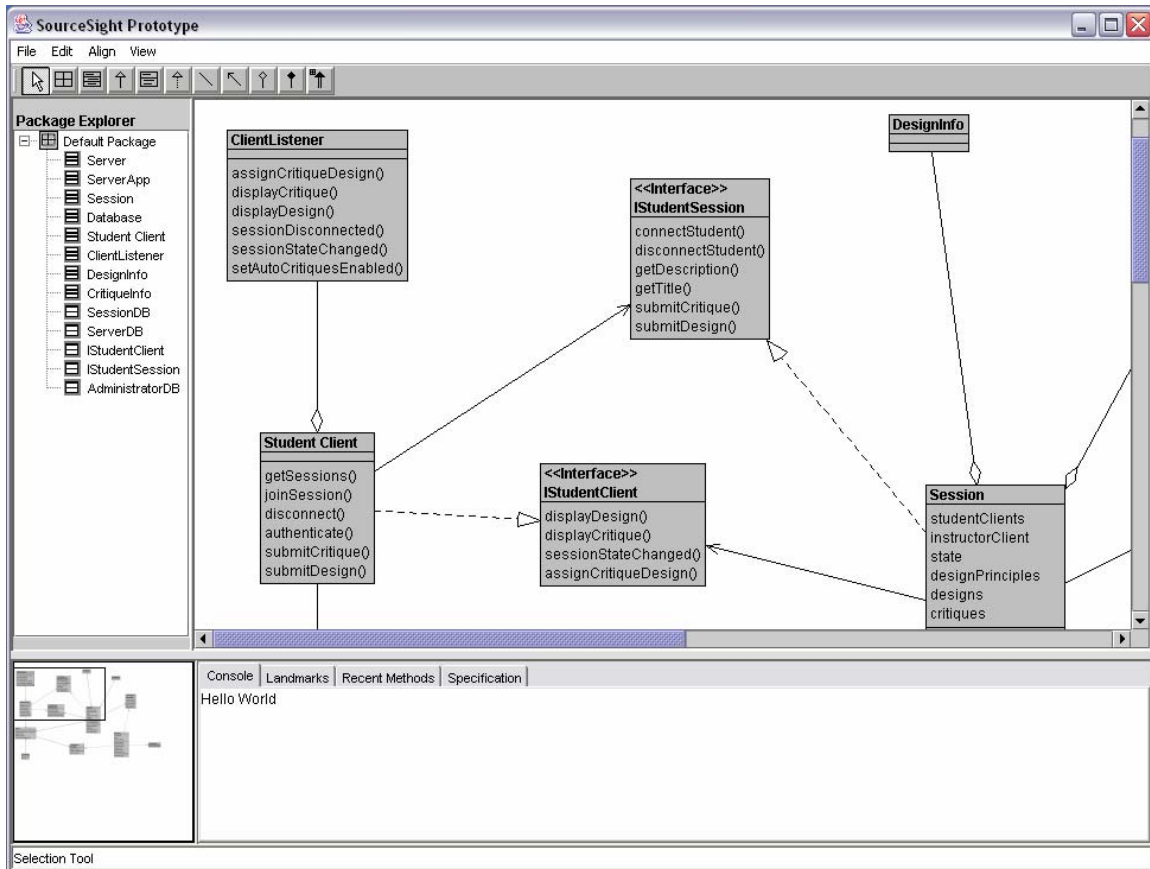


Figure 6. Hierarchical Tree View

Radar View

The Radar View is displayed in the lower left corner of the main SourceSight window (see Figure 6). The Radar View displays a miniature version of the diagram that is currently displayed in the main diagram view. This miniature allows the user to see the entire diagram regardless of how much of it is currently visible through the main view's view port (the main view port shows a portion of the diagram at a time, the user must scroll around to see other parts). The rectangle overlaid on the Radar View indicates which portion of the diagram is currently visible in the main view. The Radar View is also useful for navigation. Users can click a location in the Radar View to center the main diagram view on that location. The rectangle on the Radar View can also be dragged around to quickly pan through a large diagram.

Section 2: User Evaluation

Evaluation Protocol

Our hypothesis was that SourceSight and the class diagram based navigation will allow for better novel code understating by programmers than conventional programming tools (thus leading to better quality code changes and improved ability to locate bugs). We designed an experiment to test the hypothesis, comparing SourceSight to a commercial IDE. The experiment was also designed to provide subjective user feedback on SourceSight and its feature set.

Methods

To evaluate SourceSight, we conducted a ‘Within Subjects’ experiment comparing a functionally limited version of the Eclipse IDE with our SourceSight prototype. Each participant was asked to perform two tasks using the two programs, fill out a series of questionnaires and answer a series of interview questions. For information about the exact questions and tasks, please look at Appendix D.

Subjects

This experiment ran eight subjects aged 20-40, with seven of eight participants currently pursuing graduate studies in computer science at the University of British Columbia. Seven of the eight subjects were men. Subjects were recruited directly by the experimenters and were told that in the experiment they would be comparing file hierarchy interface and class navigation interfaces in Java. Each of these students had a moderate knowledge of UML design, and had at least one year of Java programming experience. We attempted to ensure a uniform level of knowledge and skills across subjects to reduce potential confounds to our statistics and test subject abilities that were approximately equal. The uniformity of subjects, however, implies that the claims made by this experiment cannot be generalized to all programmers. It is our belief that our subject selection does generalize to most post-graduate university students. Further, we feel that this group will be critical for future academic research and development as university community seems to have less overhead in implementing and testing new technology and systems.

Materials

A high fidelity prototype of SourceSight was designed and implemented for use in this experiment. Due to time constraints a vertical approach was implemented to allow users to parse and explore the task and training projects enough to perform the provided tasks. This limited functionality helped us to reduce bugs in the limited time frame of this experiment and spend most of our time on designing the class-oriented navigation of SourceSight, the program’s pinnacle feature.

The Eclipse IDE is a common commercial open-source Java programming environment developed by IBM. It is highly adaptable and was modified for the purposes of this experiment to provide functionality at the same level as our vertical prototype.

By implementing limited functionality for SourceSight, it would have been obvious to users that Eclipse was commercial software and thus likely the control condition. To compensate for this, and to balance the functionality of the two programs, neither program was able to compile the code in the tasks provided. Further, the Eclipse workspace was modified to remove the auto-complete, colour-coding, wizard, and search capabilities, among others. Users were then told to only use a limited subset of functions listed on the two mini-manuals. This helped make the program comparisons more focused on the code navigation tactics of the two systems. All experimental software was run on a Pentium 4 2.0 GHz laptop owned by one of the experimenters.

A Sony Digital video camera was located near the window of the office where experimentation occurred. The camera was in the room for all participants, even for subjects not being taped. During the task phase of the experiment, the camera recorded an “over the shoulder” view of the subjects (provided the subjects consented to be videotaped) so that both the side of the subject’s face and movements were recorded. This position also recorded the computer screen, although not accurately enough to read text on the screen. Camtasia, a commercial screen capturing program, was used to record a two frames per second video of the computer screen during the task phase. Both the video and Camtasia movie information was examined for qualitative information. Finally the video camera was used to record user interviews for willing participants.

Consent

Three different consent forms were used in this experiment: a questionnaire consent form, a general no-video consent form, and a general video consent form (see Appendix H for example forms). This was done to differentiate participation in various parts of the experiment. We did not require subjects to be videotaped (only 3 subjects were not videotaped) or to answer the questionnaires (all subjects consented to do this), thus separating the overall consent from the questionnaire consent seemed appropriate. Users were allowed to withdraw consent at any time during the study and their data would no longer be used.

Questionnaires

The questionnaires provided were divided into three sections: a preliminary information section, a scaled response user opinion section, and a series of long answer questions (see Appendix A for forms). The first section asks about previous programming, UML, and group development experiences. We used this data in an attempt to determine whether there were any significant similarities between the users before the experiment. This would help illuminate potential correlations between users opinions and previous experience. The second questionnaire section consisted of a series of statements which users were asked to mark on a given scale system. Questions were counter-balanced so positive and negative implications were asked. We hoped this would prevent users from quickly filling in the questionnaire without reading the questions, and would promote honest responses from users. For example, if a user selected strongly-agree for all responses in an attempt to please the experimenters, the resulting data would be neutral.

The questionnaire included the following sections:

- Questions regarding the user's agreement on a five point (strongly-disagree to strongly-agree) scale with statements comparing SourceSight and Eclipse.
- A series of statements where users were asked to state which program matched the criteria the best (the scale consisted of SourceSight, Eclipse, and No Preference). This section of the questionnaire was used to record user opinion of SourceSight in comparison with Eclipse.
- Long answer interview questions. The long answer questions about SourceSight and the experiment itself were used to gain feedback in order to improve the interface and future experimental investigations, as well as to provide some qualitative evidence regarding SourceSight's effectiveness. We asked subjects that permitted videotaping during the task phase if we could videotape their interviews as well. All of these subjects consented to this request.

Other Forms

Several other forms were used in this experiment (see Appendix G). A read-out sheet was used to describe the study, provide task instructions, and to debrief each subject in a consistent manner. This meant that each subject was given the same information verbally by the experimenters and on any forms he or she received. A general overview form was provided to each subject, which described the experiment, subject of the study, and provided contact information. The hypothesis of the study was not disclosed. Users were given minimal reference sheets for Eclipse and SourceSight, which provided the set of functions/features in each program that should be used during the experiment. These manuals also acted as tutorials for participants that needed to learn either Eclipse or SourceSight. A "UML refresher" sheet was also provided to subjects to ensure that they all had a specific base working memory of UML. A Java reference manual was provided in case any participants forgot syntax, but to our knowledge this book was not used. Experimental tasks consisted of a two steps: the addition of a class to a package based on given specifications and finding a bug within a package given a program performance description. Each experimental task was written on a separate sheet of paper and handed to a subject at the start of an experimental condition, and removed after the task period ended.

Procedure

The experiment used a balanced within subjects design to test user opinions and abilities using SourceSight compared to Eclipse. Each subject was required to perform a task in both the experimental (SourceSight) and control (Eclipse) conditions. Since subjects were acquiring knowledge of novel code, two tasks were required for each subject and thus four subject groups were required: Eclipse Task A first, Eclipse Task B first, SourceSight Task A first, and SourceSight Task B first. Due to this grouping, we used eight (a multiple of 4) subjects. All subject scores were assigned a unique identifier that was unknown to experimenters marking and analyzing the data. Each participant was tested using the following procedure:

- 1) Experimental overview and background
- 2) Sign consent forms
- 3) Software training
- 4) Perform first task

- 5) Perform second task
- 6) Fill out questionnaire section 2
- 7) Participate in interview (section 3 of the questionnaire)
- 8) Debriefing

Each subject was given a paper experimental summary and was read an experimental overview. This provided subjects with a general understanding of what they would be investigating but not why we were exploring this subject. Subjects were next asked to fill in the appropriate consent forms to begin the experiment. A preliminary questionnaire was then presented to each subject (provided they wished to fill out the questionnaire, which all of our subjects did).

Subjects were then told that they would be using Eclipse and SourceSight and asked which of these programs they were familiar with. Subjects were allocated up to fifteen minutes to explore a sample java application using SourceSight, and another 15 minutes to explore the same application using Eclipse. Further, a reference sheet for both Eclipse and SourceSight was provided at this time to allow users to explore the functionality they would be allowed. This was the only time during the experiment process that the experimenters offered advice or suggestions to the subjects. This was done to help guide and facilitate learning.

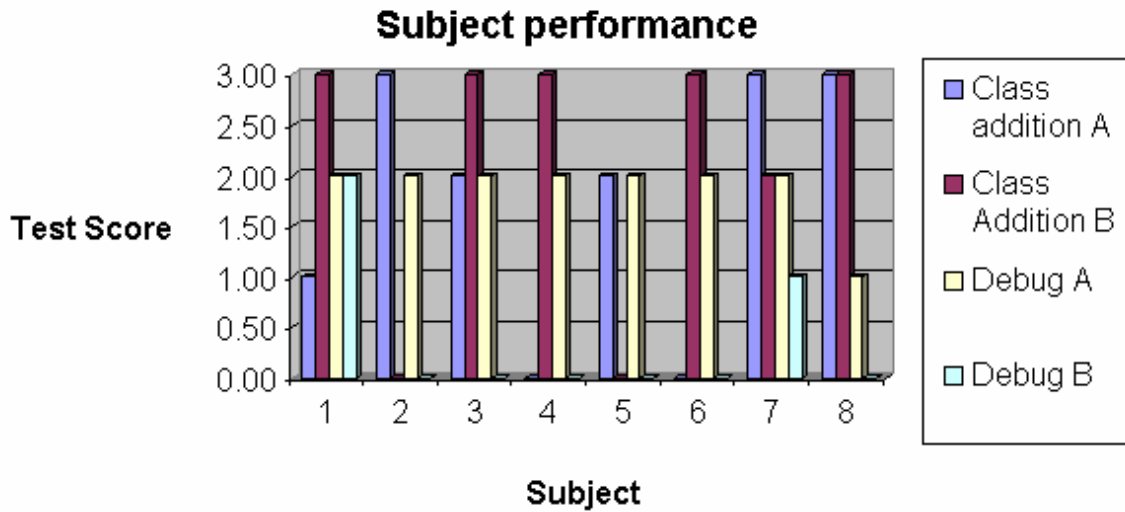
Next, the first experimental task was given to the participant. The first IDE (either Eclipse or SourceSight) was set up for the experiment, and a paper task description was given to the participant. For each task, subjects were alone in a CISCR graduate student office for up to 30 minutes or until they were finished. Experimenters waited in the hall and could be contacted by subjects by knocking on the door. After thirty minutes or when subjects were finished (whichever came first), the second task was provided to the participant and the computer was set up for the second task. Consenting subjects were filmed and had Camtasia record their screen output during these tasks. Debugging and class addition tasks were later scored based on a pre-determined marking scheme. Code additions were graded on a 0-5 scale and debugging on a scale from 0 to 2 (See Appendix G).

After the experimental tasks were finished, subjects were asked to fill out the second questionnaire (see Appendix A). Next, subjects were interviewed about their opinions about SourceSight. Consenting subjects were also filmed during this stage. Finally, when the experiment was complete, subjects were debriefed. During the debriefing, subjects were informed about the experimental hypothesis and any questions about the experiment were answered.

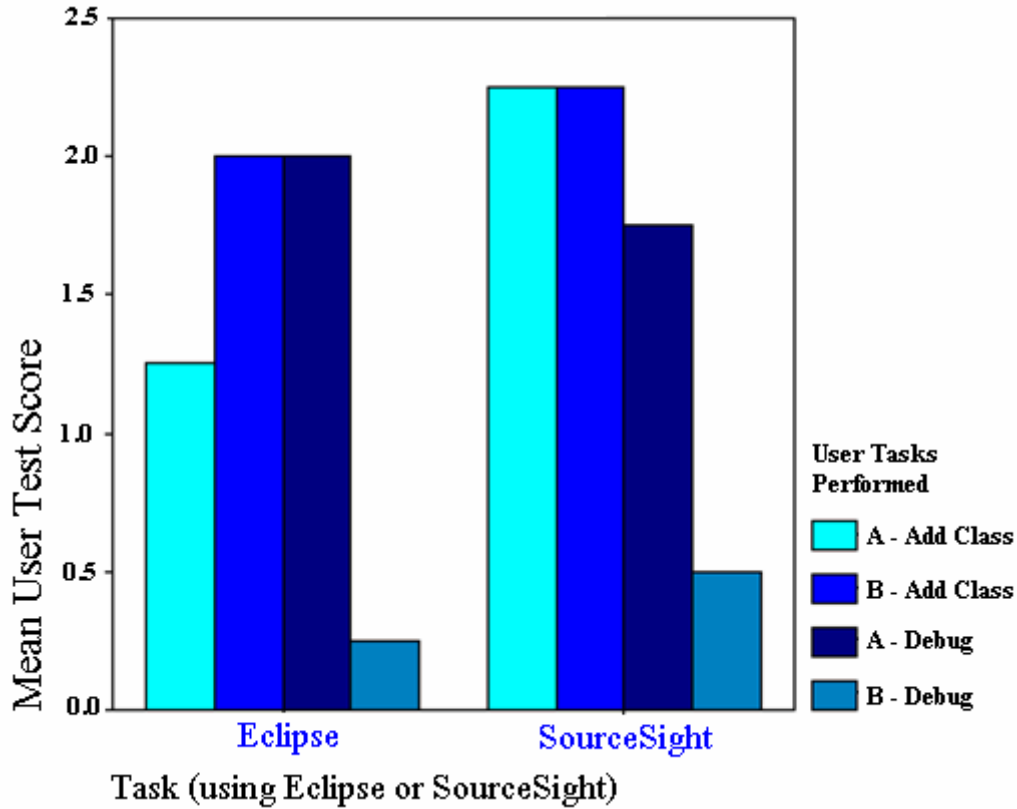
Results

Statistical analysis of user task data consisted of a 2x2 factorial ANOVA analysis with program order (SourceSight being the first or second program used) and task order as the two factors. Six dependent variables were examined: Task A class addition, Task B class addition, Task A debugging, Task B debugging, Task A's total score, and Task B's total score (see Appendix B). There was no significant main effect meaning that participants using SourceSight did not perform significantly differently than when they used Eclipse.

There was only one significant task order effect. Task A's total score was significantly better if it was the second task performed. No significant interaction effect between task order and program order were found. Although we observed a difference between the two debugging task scores, this difference was not significant (see Graph 2 below). Users tended to vary considerably with the standard deviation for both tasks being greater than 1 point on a five point marking scheme (see Graph 1 below).



Graph 1 : subject performance
Subject performance varied greatly (especially for task B)



Graph 2

Mean user test scores for Eclipse and SourceSight on each of the sub-tasks.

The chart illustrates the difference between task A and B, especially the debugging part (compare 3rd and 4th columns)

For questionnaire data, a X^2 test was performed testing against uniform random distribution of user responses. The following questions were found to be significantly non-random (the yellow highlight indicates the common selections made by users) and the p values are given in the last column (see appendix A for the full questionnaire):

SD = strongly disagree, D = disagree, N = no preference, A = Agree, SA = strongly agree

Questionnaire 1

A	The 2 tasks were equal in difficulty	<input type="checkbox"/> SD	<input checked="" type="checkbox"/> D	<input type="checkbox"/> N	<input type="checkbox"/> A	<input type="checkbox"/> SA	p = 0.0
B	Task A was more difficult than Task B	<input checked="" type="checkbox"/> SD	<input checked="" type="checkbox"/> D	<input type="checkbox"/> N	<input type="checkbox"/> A	<input type="checkbox"/> SA	p = 0.03
C	The class-based code exploration used in SourceSight was time consuming	<input checked="" type="checkbox"/> SD	<input checked="" type="checkbox"/> D	<input type="checkbox"/> N	<input type="checkbox"/> A	<input type="checkbox"/> SA	p = 0.03
D	I had greater difficulty	<input type="checkbox"/> SD	<input type="checkbox"/> D	<input type="checkbox"/> N	<input checked="" type="checkbox"/> A	<input type="checkbox"/> SA	p =

	understanding code with Eclipse than with using SourceSight						0.002
E	I believe that SourceSight would take little time to gain expert knowledge in	<input type="checkbox"/> SD	<input type="checkbox"/> D	<input type="checkbox"/> N	<input checked="" type="checkbox"/> A	<input type="checkbox"/> SA	p = 0.003
F	I think that reading other people's code was easy using SourceSight	<input type="checkbox"/> SD	<input type="checkbox"/> D	<input type="checkbox"/> N	<input checked="" type="checkbox"/> A	<input type="checkbox"/> SA	p = 0.0
G	I do not believe that SourceSight would be appropriate for a large project	<input type="checkbox"/> SD	<input checked="" type="checkbox"/> D	<input type="checkbox"/> N	<input type="checkbox"/> A	<input type="checkbox"/> SA	p = 0.003
H	The class-based code exploration used in SourceSight was unmanageable	<input checked="" type="checkbox"/> SD	<input checked="" type="checkbox"/> D	<input type="checkbox"/> N	<input type="checkbox"/> A	<input type="checkbox"/> SA	p = 0.03

Questionnaire 2

SS = SourceSight, EC = Eclipse, NP = no preference

A	This software was difficult to use	<input type="checkbox"/> SS	<input type="checkbox"/> EC	<input checked="" type="checkbox"/> NP	p = 0.04
B	This software facilitated my comprehension of novel Java code	<input checked="" type="checkbox"/> SS	<input type="checkbox"/> EC	<input type="checkbox"/> NP	p = 0.044
C	There is a steep learning curve to become an expert using this software	<input type="checkbox"/> SS	<input type="checkbox"/> EC	<input checked="" type="checkbox"/> NP	p = 0.03
D	This program made understanding overall code design easier.	<input checked="" type="checkbox"/> SS	<input type="checkbox"/> EC	<input type="checkbox"/> NP	p=0.005
E	This program increased bug creation	<input type="checkbox"/> SS	<input type="checkbox"/> EC	<input checked="" type="checkbox"/> NP	p=0.03
F	I felt in control of the interface	<input type="checkbox"/> SS	<input type="checkbox"/> EC	<input checked="" type="checkbox"/> NP	p=0.005

Finally, in the preliminary questionnaire, users frequently or always (p=0.01) sketched design diagrams before they began coding and frequently or always (p= 0.01) worked with other programmers on projects.

Discussion

From our questionnaire analysis, there seems to be several general opinions about SourceSight. Generally users seemed to appreciate the class navigation method, and believed it helped them understand code design better than using Eclipse (significant questionnaire data Q1D, Q2B, and Q2D). Further, they believed that code-based navigation was manageable (Q1H), and was not time consuming (Q1C). Finally, the majority of users believed that SourceSight is potentially scalable enough to manage large projects. Although, it should be noted, from the experimental confounds that will

be discussed shortly, most users likely knew that SourceSight was not commercial software and was not fully implemented at the time of the experiment. Thus some users may be predicting the future abilities of SourceSight rather than its current abilities. Finally, it should also be noted that users did not believe that the two tasks they performed were isomorphic; a feature we noticed during the task analysis.

The experimental task analysis showed no significant main effect between SourceSight and Eclipse, which does not support our hypothesis. It should be noted, however, that user task scores varied radically with a standard deviation of the scores for each task being greater than 1 (on a 5 point marking scale). This range of scores means that any moderate to small main effect may not be noticeable with our subject size. This range of subject ability is further supported by the lack of uniform programming experience amongst subjects collected through the preliminary questionnaire. Future studies should either involve a larger number of subjects to normalize the ability scores or recruitment of a more specific subject pool. The task analysis also demonstrated that the two subject tasks were not completely isomorphic. Although the class addition tasks seemed comparable according to their scores, the debugging tasks were not equal (although statistical analysis of this was not performed). Thus, future research should include pilot experiments to ensure that the two tasks given to subjects are truly isomorphic.

In the interviews conducted with subjects, 7 out of 8 subjects felt that viewing the class diagram and using it to navigate is helpful. They also expressed the need for adding more features to SourceSight to make it comparable to Eclipse (such as auto-indent, keyword highlighting, copy paste in diagram level etc. (these will be further discussed in the final design rational discussion).

Finally, an analysis of the Camtasia captures of subject interaction revealed that subject managed to master SourceSight in a very short time and after 3-5 minutes of work were able to work quite fluently with it. At the same time it was clear the subjects were looking for features like the ones mentioned above. It was also evident that the limited version of Eclipse was inconvenient for subjects and they were somewhat struggling with it at first.

Confounds

Several experimental confounds were noted during this experiment and may have effected our results. First, subjects were all known personally by at least one member of our team. This was predominately due to the time constraints and lack of subject compensation but such subject recruitment is still problematic. Further, three of our eight subjects are students in CPSC 544, the course for which this study was conducted. These subjects were likely to be aware of the nature of our hypothesis. Of the five remaining subjects, two subjects also explicitly knew our hypothesis. Finally, the three remaining subjects were likely able to deduce our hypothesis based on other experimental confounds and overhearing conversations our group had in public spaces. Subjects did not have a uniform knowledge of Eclipse prior to starting the experiment. This means that subjects task scores using Eclipse may have varied due to this experience (although we attempted to reduce this by training inexperienced subjects). One of our subjects was

also involved in our Phase 2 SourceSight design and this may have resulted in subject bias.

We had frequent complaints from participants that Eclipse did not work as they were accustomed to. Many users were annoyed or frustrated by the repeated error messages caused by the modified Eclipse and by many missing features like auto-complete and syntax highlighting, which they had previously used to help them program. Although our original motivation for handicapping Eclipse was to make SourceSight and Eclipse more comparable, any future experimentation should require SourceSight to gain additional functionality rather than reducing Eclipse's functionality. This would reduce user frustration, and thus reduce a potential positive bias towards SourceSight.

SourceSight not only required us to limit Eclipse's features, but the features of SourceSight that were implemented contained some bugs (see appendix E). This may have meant that, when compared to commercial software, SourceSight could be identified as a program that we designed. This in turn would allow subjects to deduce what our hypothesis was and could have lead to a Hawthorn effect, particularly with regards to our questionnaires.

Finally, the length of the experiment may have been a confound. Subjects were asked to concentrate on a series of tasks for a minimum of an hour and a half, which could have resulted in subject fatigue. Further, the experiment's length made subject recruitment problematic and subject recruitment may have been more biased due to this.

Guidelines for Future Studies

This pilot study has illuminated several important issues that we should take into account in future studies. First, future tasks should be smaller and more specific. Although this may not test all the features of SourceSight, more focused tasks would provide greater internal validity and conclusive power. The isomorphic tasks should be tested thoroughly to ensure that they are, in fact, isomorphic. Also, SourceSight should continue to be developed so that Eclipse does not need to be handicapped in future studies. Finally, we hope to make several changes to our subject recruitment and methodology. In future experiments, we would like to ensure that we have more subjects tested and their programming abilities are more uniform. Further, we would like to ensure that participants are unaware of our research, or our experimental hypothesis. This would require us to recruit subjects that do not know Eclipse, who are unaware of our research, and have approximately the same programming abilities.

Another experimental approach that could assist in verifying the value of SourceSight is conducting a lengthy in-field diary study. In such a study subjects will be given SourceSight to be used in their regular programming work for a few months and will self-report on their use of it, how useful or problematic it is etc. This will provide a more realistic evaluation of SourceSight, since short lab experiments cannot predict how useful will SourceSight be for actual work.

Final Design Rationale and Discussion

In this final section, we analyze and interpret the results obtained from the user evaluations and apply the resulting information to a short discussion on the overall quality of the system design. In doing so, we hope to verify the degree to which the objectives set in the first phase of this project have been attained.

State of the Design

SourceSight currently exists as a high-fidelity vertical prototype that focuses on supporting the navigation and visualization tasks inherent in software development. The following discussion will examine the tool within this context - meaning that only the functionalities pertaining to navigation and visualization will be examined. Features outside of this scope, such as support for compilation and debugging, will be examined and implemented in future iterations of the design.

Positive Usability Aspects of SourceSight

We begin with an examination of the usability of the tool as a whole. The applicability and interactions of each feature set chosen for implementation are directly related to this examination and will therefore be evaluated implicitly.

Visualization: SourceSight was designed to use UML, a graphical modeling technique that the target user would already be familiar with, to provide her with an abstract perspective of the system. It was our goal to remove any visible linking of the system to its underlying file structure as well as to isolate each method as an easily manageable building block of the system functionality. In doing so, we anticipate that the programmer would feel more inclined to focus on the structural concerns of the system.

The initial results from the Phase III user experiments were very positive in this regard. It will certainly take time to influence the user's perception of a system of code. However, the first step in doing so is to achieve a high degree of intuitiveness of use. Qualitative analysis was used to measure the degree of intuitiveness SourceSight provided. The results revealed that users felt that observation of the class diagram was helpful and perhaps more importantly, that SourceSight itself was relatively easy to learn.

Navigation: SourceSight was designed to use tools and navigation techniques familiar to the target user. Features such as radar views and tree hierarchies play an important role in currently available IDEs such as Eclipse and Rational XDE. However, whereas Eclipse and Rational XDE use these tools to provide the programmer with an optional and abstract perspective of the underlying file structure, it was our goal to create a system wherein the primary purpose of these tools is to support navigating the structure of the system. Specifically, the tree hierarchy is used as a means to view an outline of the classes contained within the current UML diagram and perhaps to quickly navigate to a starting point for navigation. It also facilitates adoption of SourceSight by presenting a familiar navigation aid that is common to most IDEs. The radar view, of course, is used as a mechanism to provide the user with a "birds-eye" view of the structure he is currently navigating.

Analysis of user feedback also yields positive results in this area. Quantitative analysis revealed that users felt reading other peoples code was easier using SourceSight ($p < 0.003$) and that users had greater difficulty understanding code with Eclipse than with SourceSight ($p < 0.002$). These results are very promising, as successful navigation of the system clearly has positive implications for both the intuitiveness of the visualization mechanism as well as the applicability of tool support for navigation. Observation of the users during navigation revealed that the tree/outline view was seldom used. However, this can be explained easily: the outline view was meant to provide a starting point within the diagram for a user already familiar with the system; supporting the necessary gains in efficiency as the user gains experience.

Negative Usability Aspects of SourceSight

Although the UML diagram representation of the system received positive feedback from the user evaluations, the following features require additional development.

- **Diagram Window:** The window containing the UML diagram can be panned in any direction using either the radar view or the scroll bars. However, there is currently no support for increasing or decreasing magnification of the diagram (zooming in and out). The users felt that tool support for gradual zooming would assist them in viewing tangent structural artifacts without increasing their disorientation within the diagram.
- **Method Editing:** The method-editing window contains two sub-windows: the top sub-window contains the specification for the method and the lower contains the actual method implementation. This design was chosen because the specification for each method is necessary for navigating structures without drilling down to the implementation level (hovering the mouse over a method pops up the specification). However, the users felt that this layout was confusing – they were thinking about the specification and method implementation as two separate artifacts of a method.
- **Active Class:** In updating a piece of code for the user tasks, the user would often keep two or more method editing windows open at once. Within the current version of SourceSight, however, each method window lists only the method name on the top of the window. Once multiple windows had been left open for a time, the user would often forget which class the method came from. Users suggested adding a class name as well as a method name to each method-editing window to rectify this situation.
- **Copy and Paste Coding:** The technique used to edit methods in an IDE can greatly affect the user efficiency. For example, if methods in two separate classes have similar implementations, the user will often cut and paste the code from one to the other. However, since SourceSight currently provides no support for cutting and pasting method implementations, a user who identifies two methods that have similar implementations must open up the first window, navigate to the class with a similar method and open that method-editing window, and then manually copy the code from one method to the other.

Other design flaws that were observed during user evaluations but not identified directly by the users were reflected in the previous ‘Redesign’ section.

Predicted Usability within Industry

We now apply the knowledge gained from the above analysis to a discussion on the predicted success of SourceSight in assisting the programmer in maintaining the structural integrity of his system.

Essentially, SourceSight was built to present the user with a useful abstraction of a system. The first level of the abstraction is the UML diagram view. The second layer is that the user may hover the mouse over any method or class and receive an intuitive explanation of its functionality. Only when necessary will the user be forced to deal with the actual code – and even at this time, the code will be presented in short snippets of functionality. No longer will the user be forced to search through many files, each with many lines of code, to find a bug or method to be updated.

And so the question becomes, will this abstraction succeed in creating a measurable increase in the structural integrity of the system over time?

It seems clear from the above analysis that SourceSight has great potential. There are two basic reasons that cause breakage of the design during the maintenance phase: first, in fixing a piece of code, the current programmer does not have a clear idea of what the initial programmer's intentions were. Second, the current programmer does not have time to learn the full design of a system and so decides to "blindly" update a piece of code. As described above, SourceSight provides support both for specifying the exact intentions behind a method or class and also provides a great deal of contextual information about a given class. This argument is enforced greatly by the positive user feedback regarding the readability of others' code and ease of navigation.

We believe that although there must be improvements made to the design of SourceSight in order for it to reach the usability and efficiency levels required by expert software engineers, the tool would clearly be effective at the job it was designed to perform.

Section 3: Appendices

Appendix A: Interview Questionnaires

Section 1: Preliminary User Experiences

Please fill in the appropriate bubbles or answer questions in the spaces provided.

1. For how many years have you been a computer programmer?
 - 1-3 years
 - 4-6 years
 - 7-10 years
 - more than 10 years

2. How many years of Java programming experience do you have?
 - Less than 1 year
 - 1-2 years
 - 3-4 years
 - 5 or more years

3. Do you design algorithms and programs by sketching design diagrams before you begin coding?
 - Never (0% of the time)
 - Rarely (~25%)
 - Occasionally (~50%)
 - Frequently (~75%)
 - Always (~100%)

4. How frequently have do you use UML to design programs?
 - Never (0% of the time)
 - Rarely (~25%)
 - Occasionally (~50%)
 - Frequently (~75%)
 - Always (~100%)

5. How often do you work with other programmers on a project?
 - Never (0% of the time)
 - Rarely (~25%)
 - Occasionally (~50%)
 - Frequently (~75%)
 - Always (~100%)

Section 2: Comparing Eclipse and SourceSight

With respect to using SourceSight and Eclipse, please indicate the extent to which you agree or disagree with the following statements:

SD = Strongly Disagree
 D = Disagree
 N = Neutral
 A = Agree
 SA = Strongly Agree

The 2 tasks were equal in difficulty	<input type="checkbox"/> SD	<input type="checkbox"/> D	<input type="checkbox"/> N	<input type="checkbox"/> A	<input type="checkbox"/> SA
Task A was more difficult than Task B	<input type="checkbox"/> SD	<input type="checkbox"/> D	<input type="checkbox"/> N	<input type="checkbox"/> A	<input type="checkbox"/> SA
SourceSight provided more information than Eclipse	<input type="checkbox"/> SD	<input type="checkbox"/> D	<input type="checkbox"/> N	<input type="checkbox"/> A	<input type="checkbox"/> SA
SourceSight did not help me understand the code better	<input type="checkbox"/> SD	<input type="checkbox"/> D	<input type="checkbox"/> N	<input type="checkbox"/> A	<input type="checkbox"/> SA
SourceSight's interface was easy to understand	<input type="checkbox"/> SD	<input type="checkbox"/> D	<input type="checkbox"/> N	<input type="checkbox"/> A	<input type="checkbox"/> SA
The class-based code exploration used in SourceSight was time consuming	<input type="checkbox"/> SD	<input type="checkbox"/> D	<input type="checkbox"/> N	<input type="checkbox"/> A	<input type="checkbox"/> SA
The radar view helped me explore the project efficiently	<input type="checkbox"/> SD	<input type="checkbox"/> D	<input type="checkbox"/> N	<input type="checkbox"/> A	<input type="checkbox"/> SA
The multiple windows required to explore code in SourceSight is awkward	<input type="checkbox"/> SD	<input type="checkbox"/> D	<input type="checkbox"/> N	<input type="checkbox"/> A	<input type="checkbox"/> SA
I had greater difficulty understanding code with Eclipse than with using SourceSight	<input type="checkbox"/> SD	<input type="checkbox"/> D	<input type="checkbox"/> N	<input type="checkbox"/> A	<input type="checkbox"/> SA
It took me more time to make changes using Eclipse than using SourceSight	<input type="checkbox"/> SD	<input type="checkbox"/> D	<input type="checkbox"/> N	<input type="checkbox"/> A	<input type="checkbox"/> SA
I believe that SourceSight would take little time to gain expert knowledge in	<input type="checkbox"/> SD	<input type="checkbox"/> D	<input type="checkbox"/> N	<input type="checkbox"/> A	<input type="checkbox"/> SA
I think that reading other people's code was easy using SourceSight	<input type="checkbox"/> SD	<input type="checkbox"/> D	<input type="checkbox"/> N	<input type="checkbox"/> A	<input type="checkbox"/> SA
I do not believe that SourceSight would be appropriate for a large project	<input type="checkbox"/> SD	<input type="checkbox"/> D	<input type="checkbox"/> N	<input type="checkbox"/> A	<input type="checkbox"/> SA
The class-based code exploration used in SourceSight was unmanageable	<input type="checkbox"/> SD	<input type="checkbox"/> D	<input type="checkbox"/> N	<input type="checkbox"/> A	<input type="checkbox"/> SA

There are a number of criteria listed below. Please select which programming environment would be your 1st choice according to each of the criteria. If you really cannot make a choice for a given criteria please select “No Preference”

SS = SourceSight EC = Eclipse IDE NP = No Preference
--

Criteria	1 st Choice		
	<input type="checkbox"/> SS	<input type="checkbox"/> EC	<input type="checkbox"/> NP
This software was difficult to use	<input type="checkbox"/> SS	<input type="checkbox"/> EC	<input type="checkbox"/> NP
This software helped me program more efficiently	<input type="checkbox"/> SS	<input type="checkbox"/> EC	<input type="checkbox"/> NP
This software facilitated my comprehension of novel Java code	<input type="checkbox"/> SS	<input type="checkbox"/> EC	<input type="checkbox"/> NP
There is a steep learning curve to become an expert using this software	<input type="checkbox"/> SS	<input type="checkbox"/> EC	<input type="checkbox"/> NP
This program made understanding overall code design easier.	<input type="checkbox"/> SS	<input type="checkbox"/> EC	<input type="checkbox"/> NP
This program increased bug creation	<input type="checkbox"/> SS	<input type="checkbox"/> EC	<input type="checkbox"/> NP
Code navigation and exploration was easy using this program	<input type="checkbox"/> SS	<input type="checkbox"/> EC	<input type="checkbox"/> NP
I sometimes felt “lost” in the code and was unsure what class I was investigating.	<input type="checkbox"/> SS	<input type="checkbox"/> EC	<input type="checkbox"/> NP
I felt in control of the interface	<input type="checkbox"/> SS	<input type="checkbox"/> EC	<input type="checkbox"/> NP
The software required too much clicking and interaction	<input type="checkbox"/> SS	<input type="checkbox"/> EC	<input type="checkbox"/> NP
This software is satisfying to use	<input type="checkbox"/> SS	<input type="checkbox"/> EC	<input type="checkbox"/> NP

Appendix B: Data Analysis

Univariate Analysis of Variance (Task A)

		Notes
Output Created		06-DEC-2003 15:16:55
Comments		
Input	Data	C:\Users\David Temp\ExpData12503.sav
	Filter	<none>
	Weight	<none>
	Split File	<none>
	N of Rows in Working Data File	8
Missing Value Handling	Definition of Missing	User-defined missing values are treated as missing.
	Cases Used	Statistics are based on all cases with valid data for all variables in the model.
Syntax		UNIANOVA task_a BY ta_ecl ta_first /METHOD = SSTYPE(3) /INTERCEPT = INCLUDE /PRINT = DESCRIPTIVE ETASQ HOMOGENEITY /CRITERIA = ALPHA(.05) /DESIGN = ta_ecl ta_first ta_ecl*ta_first .
Resources	Elapsed Time	0:00:00.21

Between-Subjects Factors		
		N
TA_ECL	0	4
	1	4
TA_FIRST	.00	4
	1.00	4

Descriptive Statistics				
Dependent Variable: TASK_A				
TA_ECL	TA_FIRST	Mean	Std. Deviation	N
0	.00	2.000	.0000	2
	1.00	4.500	.7071	2
	Total	3.250	1.5000	4
1	.00	3.500	.7071	2
	1.00	4.500	.7071	2

	Total	4.000	.8165	4
Total	.00	2.750	.9574	4
	1.00	4.500	.5774	4
	Total	3.625	1.1877	8

Levene's Test of Equality of Error Variances(a)			
Dependent Variable: TASK_A			
F	df1	df2	Sig.
.	3	4	.
Tests the null hypothesis that the error variance of the dependent variable is equal across groups.			
a Design: Intercept+TA_ECL+TA_FIRST+TA_ECL * TA_FIRST			

Tests of Between-Subjects Effects						
Dependent Variable: TASK_A						
Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	8.375(a)	3	2.792	7.444	.041	.848
Intercept	105.125	1	105.125	280.333	.000	.986
TA_ECL	1.125	1	1.125	3.000	.158	.429
TA_FIRST	6.125	1	6.125	16.333	.016	.803
TA_ECL * TA_FIRST	1.125	1	1.125	3.000	.158	.429
Error	1.500	4	.375			
Total	115.000	8				
Corrected Total	9.875	7				
a R Squared = .848 (Adjusted R Squared = .734)						

Univariate Analysis of Variance (Task B)

Notes		
Output Created	06-DEC-2003 15:21:03	
Comments		
Input	Data	C:\Users\David Temp\ExpData12503.sav
	Filter	<none>
	Weight	<none>
	Split File	<none>
	N of Rows in Working Data File	8
Missing Value Handling	Definition of Missing	User-defined missing values are treated as missing.
	Cases Used	Statistics are based on all cases with valid data for all variables in the model.

Syntax	UNIANOVA task_b BY ta_ecl ta_first /METHOD = SSTYPE(3) /INTERCEPT = INCLUDE /EMMEANS = TABLES(ta_ecl) /EMMEANS = TABLES(ta_first) /EMMEANS = TABLES(ta_ecl*ta_first) /PRINT = DESCRIPTIVE ETASQ HOMOGENEITY /CRITERIA = ALPHA(.05) /DESIGN = ta_ecl ta_first ta_ecl*ta_first .
Resources	Elapsed Time 0:00:00.04

Between-Subjects Factors		
		N
TA_ECL	0	4
	1	4
TA_FIRST	.00	4
	1.00	4

Descriptive Statistics				
Dependent Variable: TASK_B				
TA_ECL	TA_FIRST	Mean	Std. Deviation	N
0	.00	3.0000	.00000	2
	1.00	2.0000	1.41421	2
	Total	2.5000	1.00000	4
1	.00	3.5000	.70711	2
	1.00	2.5000	2.12132	2
	Total	3.0000	1.41421	4
Total	.00	3.2500	.50000	4
	1.00	2.2500	1.50000	4
	Total	2.7500	1.16496	8

Levene's Test of Equality of Error Variances(a)			
Dependent Variable: TASK_B			
F	df1	df2	Sig.
.	3	4	.
Tests the null hypothesis that the error variance of the dependent variable is equal across groups.			
a Design: Intercept+TA_ECL+TA_FIRST+TA_ECL * TA_FIRST			

Tests of Between-Subjects Effects						
Dependent Variable: TASK_B						
Source	Type III Sum of	df	Mean	F	Sig.	Partial Eta

	Squares		Square			Squared
Corrected Model	2.500(a)	3	.833	.476	.716	.263
Intercept	60.500	1	60.500	34.571	.004	.896
TA_ECL	.500	1	.500	.286	.621	.067
TA_FIRST	2.000	1	2.000	1.143	.345	.222
TA_ECL * TA_FIRST	.000	1	.000	.000	1.000	.000
Error	7.000	4	1.750			
Total	70.000	8				
Corrected Total	9.500	7				

a R Squared = .263 (Adjusted R Squared = -.289)

Estimated Marginal Means

1. TA_ECL				
Dependent Variable: TASK_B				
TA_ECL	Mean	Std. Error	95% Confidence Interval	
			Lower Bound	Upper Bound
0	2.500	.661	.664	4.336
1	3.000	.661	1.164	4.836

2. TA_FIRST				
Dependent Variable: TASK_B				
TA_FIRST	Mean	Std. Error	95% Confidence Interval	
			Lower Bound	Upper Bound
.00	3.250	.661	1.414	5.086
1.00	2.250	.661	.414	4.086

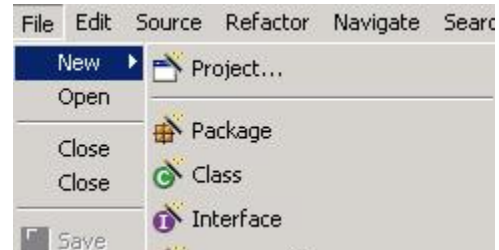
3. TA_ECL * TA_FIRST					
Dependent Variable: TASK_B					
TA_ECL	TA_FIRST	Mean	Std. Error	95% Confidence Interval	
				Lower Bound	Upper Bound
0	.00	3.000	.935	.403	5.597
	1.00	2.000	.935	-.597	4.597
1	.00	3.500	.935	.903	6.097
	1.00	2.500	.935	-.097	5.097

Appendix C: Reference Sheets

Eclipse (Modified) Quick Reference

Open an existing project

Projects should already be loaded into the workspace. Additional files can be opened using File | Import or File | Open External File



Create a new project

File | New | Project

Save a project

File | Save or Ctrl + S

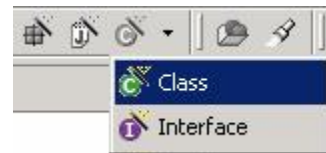
Create a new Package

File | New | Package or New Package Toolbar button



Create a new Interface

File | New | Interface or New Class | Interface Toolbar button



Create a new Class

File | New | Class or New Class | Class Toolbar button

Add new Method for a Class

Edit the source code of desired class and type in new method details.

Add new Attribute for a Class

Edit the source code of desired class and type in new attribute details.

Delete existing Method from a Class

Edit the method (See Edit Method Source) and remove all method text in the editor window or Right Click desired method in Package Navigator and select Delete or press Delete key on keyboard when method is selected.

Delete existing Class, Package or Interface

Right click desired object in Package Navigator and select Delete or press Delete key on keyboard when item is selected.



Edit Class Name

Edit the class and change the class name OR Right click desired class in Package Navigator and select Refactor | Rename.



Edit Method name

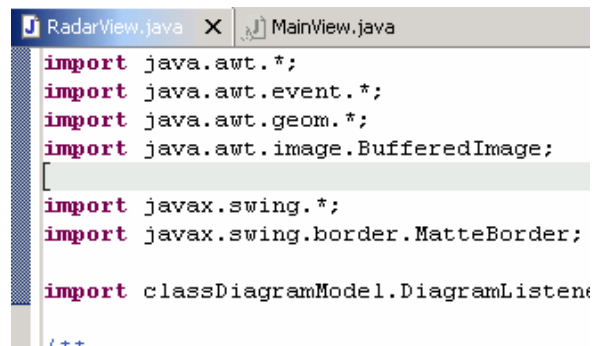
Edit the class and change the method name OR Right click desired method in Package Navigator and select Refactor | Rename.

Edit Package name

Right click desired package in the Package Navigator and select Refactor | Rename.

Edit Method or Class Source Code or Specification

Double click on desired class or method in the Package Navigator and edit desired code or comments in the editor window.



```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.awt.image.BufferedImage;
[
import javax.swing.*;
import javax.swing.border.MatteBorder;

import classDiagramModel.DiagramListene
/++
```

View Method Specification

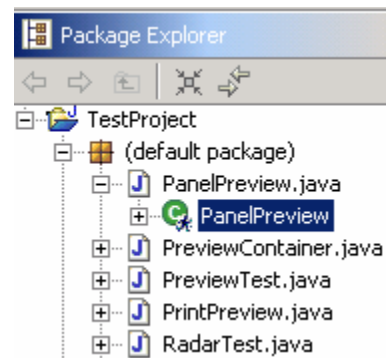
Double click on the desired method in the Package Navigator and view the specification in the Editor window.

Hide / Show Package Explorer

Right click on any classes tab in the code editor window and select Maximize (to Hide) or Restore (to Show) or Double Click tab to switch between Show and Hide.

Expand / Navigate into package

Single click on the expansion (+) symbol in the Package Navigator next to the desired package.



Collapse / Navigate out of package

Single click on the collapse (-) symbol in the Package Navigator next to the desired package.

SourceSight Quick Reference

Open an existing project

File | Open Design *or* Ctrl + O

Create a new project

File | Create New Design *or* Ctrl + N

Save a project

File | Save *or* Ctrl + S



Create a new Package

Press the New Package button and click on the UML diagram



Create a new Interface

Press the New Interface button and click on the UML diagram



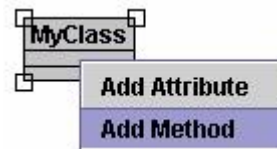
Create a new Class

Press the New Class button and click on the UML diagram



Add new Method for a Class

Right click on Class icon and select Add Method



Add new Attribute for a Class

Right click on Class icon and select Add Attribute

Delete existing Method from a Class

Edit the method (See Edit Method Source) and remove all text in the editor window.

Delete existing Class, Package or Interface

Highlight Item to delete and press Delete key on Keyboard.

Edit Class Name

Double-click Class to edit, then Click anywhere else on diagram to finish.



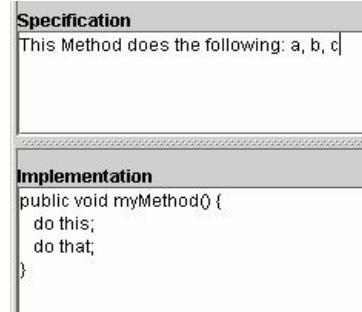
Edit Method name

Edit the method Source and change the name.

(See Edit Method Source)

Edit Method Source Code or Specification

Double click when mouse pointer is above the desired method. Edit desired code or comments and click Close to save.



Edit Package name

Currently unavailable

Add UML Annotation

See UML Reference Sheet for details



Navigate into package (down one level)

Double click on the desired package icon in the diagram

Navigate out of package (up one level)

Click on the To Parent Diagram toolbar button



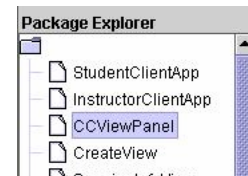
Navigate UML Diagram

Use the radar view in the bottom left corner or the scrollbars around the diagram.



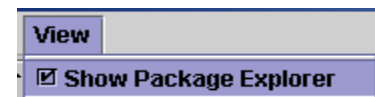
Centre UML Diagram on a Class

Highlight the desired class in the Package Explorer



Hide / Show Package Explorer

Check or Uncheck box under View | Show Package Explorer

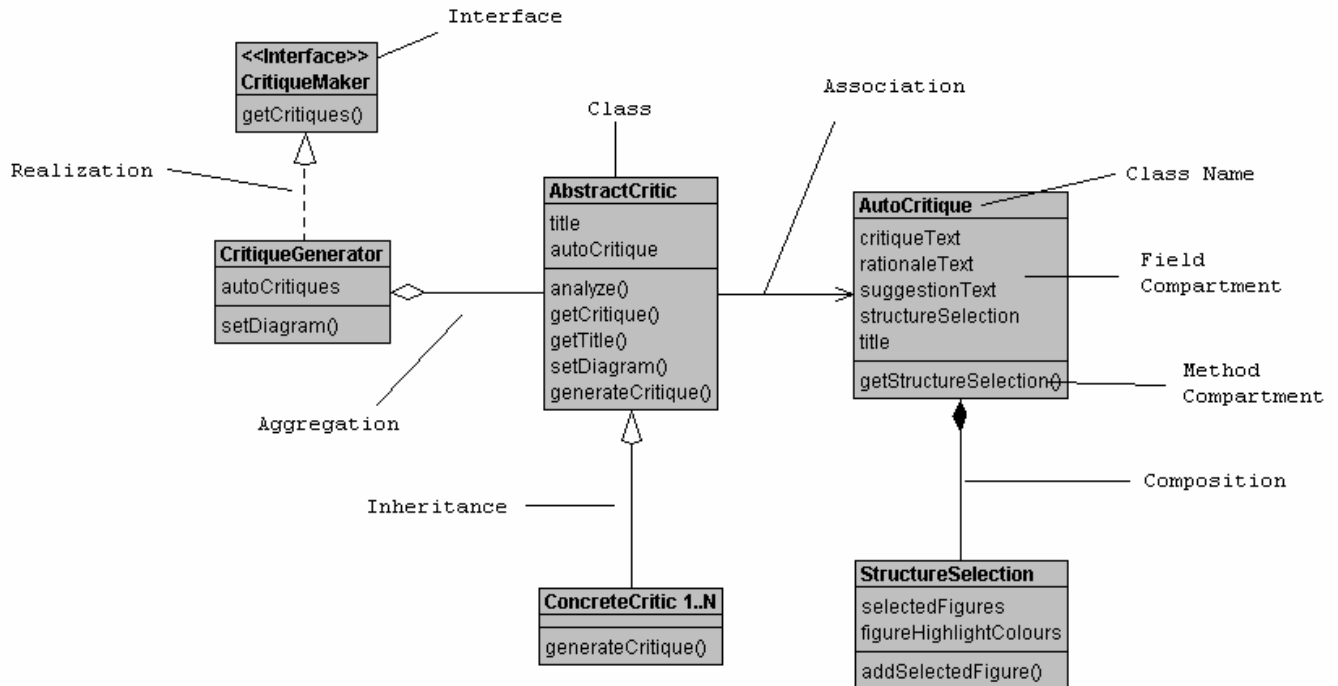


View Method Specification

Place the mouse pointer over the selected method on the diagram. The specification will appear as a Tooltip, and also in the bottom window if the Specification tab is selected.



SourceSight UML Reference



The subset of UML notation used in SourceSight consists of the labeled elements above.

Components

- **Class.** The top compartment contains the class name, the middle compartment contains fields that belong to the class, and the bottom compartment contains its methods.
- **Interface.** Interfaces have the <<Interface>> stereotype above their name that distinguishes them from classes. Interfaces do not have a compartment for fields.

Connectors

- **Inheritance.** Declares a superclass-subclass relationship. In the diagram above, ConcreteCritic inherits from AbstractCritic.
- **Realization.** Declares that a class realizes an interface (implements all the methods declared in the interface). In the diagram above, CritiqueGenerator realizes the CritiqueMaker interface.
- **Association.** This connector indicates that one class references another. In the diagram above, AbstractCritic has a reference to AutoCritique.
- **Aggregation.** This open-diamond connector indicates that one class aggregates, or contains a collection of, other classes. The diamond decoration appears on the aggregator's side. In the example above, CritiqueGenerator aggregates instances of AbstractCritic.

- **Composition.** A solid-diamond composition relationship indicates that one class is composed of another class. This declares a part-whole relationship in which both classes are created and destroyed together. In the example above, a StructureSelection is part of an AutoCritique.

Appendix D: User Introduction

Read Out Instructions

Experimental Overview

We are attempting to explore the idea that programmers design systems and find methods of explaining a software system without the need for programming language syntax. We would like to investigate the effects of integrating UML design diagrams into java projects when programming. To accomplish this, we are testing two programs, SourceSight and Eclipse, to determine how users explore new code and if they would prefer to examine code using a file hierarchy tree view, or using UML class diagram information. We are also investigating the personal opinions of users using the two systems. Since the two programs have different set of functions, we ask you to use the limited subset of functions discussed on the two “mini-manuals” provided. You will investigate two novel pieces of code, one using Eclipse and one using SourceSight and you will be asked to add functionality and debug the code. We are attempting to investigate the differences between two programming interfaces and not testing your ability to program.

The provided overview sheet will explain the tasks you will be involved in during this study. If at any time you have any questions please ask an experimenter. If the experimenter is outside the room, please knock on the door. He is waiting outside in the hall. If at any time you wish to leave the study, please tell an experimenter. Please note that videotaping will be conducted during the task oriented testing phase. If you do not wish to be filmed, please tell an experimenter. Please take a moment to read the overview. Also, please take a minute to sign the appropriate consent forms. If you would like to know how you did, we can supply that information at the end of the experiment. Do you have any questions?

Time Frame:

Task	Estimated Time Required (minutes)
Introduction and Overview	5
Source Sight Training	15
Eclipse Training	15 (optional)
Package 1 Task	30
Package 1 Questionnaire	0
Package 2 Task	30
Package 2 Questionnaire	0
Final Questionnaire	15
Debriefing	5
Setup Time & Miscellaneous	10
Overall	110-125 minutes (maximum)

Source Sight/Eclipse: If you are not familiar with this program, you will be allocated up to 15 minutes to explore a test project. When you feel comfortable with the software, please let an experimenter know.

Note that for Eclipse, implementing a new class requires you to create a new .java file and to write stub code in it. For SourceSight, implementing a new class requires you to add an object to the UML diagram, add appropriate connections in the diagram, and add method stubs. Please refer to the Java manual provided if you have any problems. (constructors can be included. Arrays and vectors are shown by the aggregation symbol). If you notice any bugs or glitches in either program, please try to ignore them if possible. If you are unable to ignore the error (crash bug, etc) please inform the experimenter. After the experiment we would appreciate knowing what bugs were discovered.

For this task, you will be given 30 minutes to complete the task provided. If you have any questions, please knock on the door to alert an experimenter.

Answer the question as well as you can and inform us of your decisions at the end of the task.

Questionnaire: We would now like you to fill out the following questionnaires and answer several questions. The purpose of the questionnaire is to explore your personal opinions about the two programs. VIDEO: We would also like the interview questions to be taped. Is this ok?

Debriefing: The goal of this experiment was to see if using the UML class navigation system to program SourceSight improved a programmer's ability to navigate and understand novel code. Specifically, we wanted to ensure that bugs could easily be identified using SourceSight and we hoped that SourceSight would facilitate debugging better than Eclipse. We also wanted to see if it would be simpler for users to understand complex object oriented UML structures and be able to efficiently add a class without breaking the program design. Finally, we wished to demonstrate that SourceSight does not perform significantly worse than Eclipse in user opinions about file exploration and interface design. Do you have any questions? Thank you. Would you like to be informed about the final results of this experiment?

Experimental Overview

We are attempting to explore the idea that programmers design systems (using formal UML tools, doodles on napkins, etc) and find methods of explaining a software system without the need for programming language syntax. We find it surprising that few or no IDEs explicitly incorporate these designs with the code. This would ensure that subsequent users of a system would see the structure of the system the same way the designer did. We are hoping to investigate what effects requiring the user to view the associated UML diagram have on program comprehension and programming ability. Further, we want to collect user subjective opinions about this methodology.

Methodology

As a subject in our study, you will be required to examine two (2) java packages using a different programming environment each time. Since the features provided by the eclipse IDE and SourceSight are different, we would appreciate it if you only used functions and features presented on the two mini-manuals provided. For each package you explore, you will be given twenty (20) minutes and provided a task list to perform on the code. It is ok if you are unable to complete all tasks in the time provided, but we would suggest that for any task you decide to perform, that you perform the task to completion. Please note that we do not want you to compile and run the code, but rather just modify the syntax. **If you have any questions or problems during this time, please feel free to knock on the door to talk with an experimenter.** Each time you have completed the task list or twenty minutes have passed, you will be provided a questionnaire. After both packages have been examined, you will be asked to fill out a final questionnaire about your preferences and opinions about the programs.

Time Frame:

Task	Estimated Time Required (minutes)
Introduction and Overview	5
SourceSight Training	15
Eclipse Training	15 (optional)
Package 1 Task	30
Package 1 Questionnaire	5
Package 2 Task	30
Package 2 Questionnaire	
Final Questionnaire	15
Debriefing	5
Setup Time & Miscellaneous	10
Overall	110-125 minutes (maximum)

For those subjects who agree to be videotaped, we would greatly appreciate further information about our experiment and the two programs. We would appreciate it if you permitted us to videotape the interview/questionnaire at the end of the experiment. If you are willing to participate in a taped interview please inform one of the experimenters.

Filming

In addition to your questionnaire answers and your performance on the various tasks, we hoped to collect relevant user data using videotaping. During the two programming tasks you will be given, a video camera will be recording facial reactions. We would appreciate it if you did not modify your traditional behavior and reactions due to the video camera. In addition to video taping users, we will be using Camtasia to record screen information during the tasks. Permitting video camera and screen capturing data collection is not mandatory for your participation in this experiment. If you do not wish to be filmed, or if you do not want your screen information collected, please tell an experimenter and video will not be used.

The Eclipse IDE

One of the programs used to explore this experiment's packages is the Eclipse IDE. You will be provided with a small guide/manual to use this program. If you are not familiar with this IDE, please inform the experimenters. We will provide you with some basic functionality on how to use the program and will allow you fifteen minutes with a test package to become familiar with the interface.

Source Sight:

The other program used to explore this experiment's packages is SourceSight. SourceSight uses UML design diagrams to parse and explore java code. You will be provided with a small guide/manual to use this program. We will then provide you with some basic functionality on how to use the program and will allow you ten minutes with a test package to become familiar with the interface.

Leaving the Experiment

If at any time during the experiment you do not want to continue, please inform an experimenter and your data will not longer be used. If you have any questions after the experiment has been completed, please contact David Sprague at (604) 872-6911 or dsprague@cs.ubc.ca. Also, upon request, we can provide you with the final experimental results of this study. Thank you once again for your participation.

Appendix E: SourceSight Bug List

- The '<' , '>' symbols did not appear correctly in the code, when opening a method edit window (due to XML problems). This made subjects wonder if that was the bug and distracted them from the main search course.
- Radar view occasionally failed to display properly
- The position of the right click menu was sometimes dependent on the object's position in the UML diagram. The menu could sometimes appear off the screen if you were at the bottom of a UML diagram, so subjects were unable to use it, unless they dragged the class up the screen
- Sometimes when clicking on the diagram canvas the previously selected class would jump to that position and thus changing the diagram layout

Appendix F: External Media

Please see attached CD for additional data.

The CD contains:

- The presentation given in class
- The video shown in class, presenting SourceSight, the iterative design process and the evaluation
- The full data collected in the experiment
- All forms and documents used in the experiment
- SourceSight program source code

Appendix G: Evaluation Tasks

Task A

Task Overview

In this task you will be presented with an existing project in an Integrated Development Environment (IDE). Your task is to understand the nature of the source code in order to extend the program in a way that is consistent with the existing design.

Existing Code Overview

The code loaded into the IDE implements a model of a simple UML class diagram. The diagram modeled consists of classes that are connected by various types of connection relationships.

Task Description

An interface in Java is a construct that is similar to an abstract class in C++. Interfaces define behaviour and therefore methods, but cannot be instantiated and do not have instance data. Your task is to add support for Java interfaces to the program that has been loaded into the IDE. Users should be able to define methods that belong to the interface but interfaces cannot connect to other elements of the design. Interfaces should also have names, and xy coordinates. The program should also ensure that all interfaces have a unique identifier.

Bug Description

When used by a driver program, objects that represent UML classes are created and added to the diagram. However, when a collection of classes is retrieved from the diagram the newly added class is not returned. Please identify line(s) of code that might be causing this problem.

Task A Solution

A possible solution for Task A includes the following modifications:

- Addition of a new class to represent the Interface
- Interface should extend UMLComponent
- Interface should aggregate a collection of methods using the existing Method object
- The interface constructor should call the superclass constructor
- A method for adding Methods to the interface should be defined
- Something should be added to the Diagram.addComponent() method to make sure that the unique ID is set.

Bonus Points

- removeMethod() and getMethods() methods consistent with the UMLClass class
- Calls to designChanged() in addMethod() and removeMethod()

Negative Points

- Wrong class is extended
- -1 for each case where the subject reimplements a method that is already defined in the system.
- -1 for each case where a variable defined by a superclass is needlessly redefined in a subclass

Bug location:

Code was commented out on ClassDiagram::addComponent

Marking scheme for bug finding task

0 - Did not find bug

1 - On the right track

2 - Successfully identified the bug

Task B

Task Overview

In this task you will be presented with an existing project in an Integrated Development Environment (IDE). Your task is to understand the nature of the source code in order to extend the program in a way that is consistent with the existing design. You will also be asked to find the source of a bug.

Existing Code Overview

The code loaded into the IDE is the server part of a distributed client/server system that allows users to design and swap class diagram sketches.

Task Description

The current system allows students to connect to a session on the server to submit and share designs. Instructors have requested that a new type of user be added to the system to allow additional administrative privileges. Your task is to extend the system so that an Instructor Client object can be used instead of a Student Client object to access the server. The Instructor Client works exactly like a Student Client except that the authentication and the routine for actually joining a session will be slightly different.

Please add a class to support this functionality. You can declare methods and fields/attributes but do not actually implement any of the code.

Bug Description

Clients are able to use the server to submit diagrams that they have created. However, there are problems when designs are swapped between the different clients. The server seems to crash immediately after swapping the users' designs. Please identify the line(s) of code that might be causing this problem.

Task B Solution

A possible solution for Task B includes the following modifications:

- Addition of a new class called InstructorClient to represent the new user
- InstructorClient should extend StudentClient
- InstructorClient should define methods that override joinSession() and authenticate() implementations in StudentClient

Negative Points

- Wrong class is extended
- -1 for each case where the subject reimplements a method that is already defined in the system.

- -1 for each case where a variable defined by a superclass is needlessly redefined in a subclass

Bug Location

The index of iteration in `Session.shuffleDesigns()` is incorrect.

Marking scheme for bug finding task

0 - Did not find bug

1 - On the right track

2 - Successfully identified the bug

Appendix H: Sample Consent Forms

Sample Form One



THE UNIVERSITY OF BRITISH COLUMBIA

Department of Computer Science
2366 Main Mall
Vancouver, B.C., V6T 1Z4

September 1, 2003

Sample Consent Form (videotaping included)

Human-Computer Interaction Course Projects (CPSC 444/544)

Principal Investigator

Dr. Joanna McGenere, Assistant Professor, Department of Computer Science, University of British Columbia (604) 827-5201

Student Investigators

Lior Berry	(604) 307-2479
Wesley Coelho	(604) 739-2052
Edward McCormick	(604) 228-0909
David Sprague	(604) 872-6911
Trevor Young	(604) 872-6087

Project Purpose and Procedures

This course project is designed to investigate how people interact with certain types of interactive technology. Interactive technology includes applications that run on a standard desktop or laptop computer, such as a word processor, web browser, and email, as well as applications on handheld technology, such as the datebook on the Pocket PC, and also applications on more novel platforms such a SmartBoard (electronic whiteboard) or a Diamond Touch tabletop display.

The purpose of this course project is to gather information that can help improve the design of interactive technology. You will be asked to use one or more forms of interactive technology to perform a number of tasks. We will observe you performing those tasks and analyze how the technology is used. You may be asked to complete a number of questionnaires and we may ask to interview you to find out your impressions of the technology. You will be asked to participate in at most 3 sessions, each lasting no

more than 1 hour. The sessions may also be videotaped. Videotapes will be used for analysis and may also be used for class project presentations and other research presentations in the Department of Computer Science at the University of British Columbia. You have the option not to be videotaped.

Although only a course project in its current form, this project may, at a later date, be extended by one or more of the student investigators to form the basis of his/her thesis research.

Confidentiality

The identities of all people who participate will remain anonymous and will be kept confidential. The one exception is that excerpts from the videotape may be presented as described above, and your identity may be revealed through those video excerpts. Identifiable data and videotapes will be stored securely in a locked metal filing cabinet or in a password protected computer account. All data from individual participants will be coded so that their anonymity will be protected in any reports, research papers, thesis documents, and presentations that result from this work.

Remuneration/Compensation

We are very grateful for your participation. However, you will not receive compensation of any kind for participating in this project.

Contact Information About the Project

If you have any questions or require further information about the project you may contact **David Sprague (604) 872-6911**

Contact for information about the rights of research subjects

If you have any concerns about your treatment or rights as a research subject, you may contact the Research Subject Information Line in the UBC Office of Research Services at 604-822-8598.

Consent

We intend for your participation in this project to be pleasant and stress-free. Your participation is entirely voluntary and you may refuse to participate or withdraw from the study at any time.

Your signature below indicates that you have received a copy of this consent form for your own records.

Your signature indicates that you consent to participate in this project. You do not waive any legal rights by signing this consent form.

I, _____, agree to participate in the project as outlined above. My participation in this project is voluntary and I understand that I may withdraw at any time.

Participant's Signature Date

Student Investigator's Signature Date

Sample Form Two



THE UNIVERSITY OF BRITISH COLUMBIA

Department of Computer Science
2366 Main Mall
Vancouver, B.C., V6T 1Z4

September 1, 2003

Sample Consent Form (no videotaping)

Human-Computer Interaction Course Projects (CPSC 444/544)

Principal Investigator

Dr. Joanna McGenere, Assistant Professor, Department of Computer Science, University of British Columbia (604) 827-5201

Co-Investigator

Dr. Brian Fisher, Adjunct Professor, Department of Computer Science, University of British Columbia (604) 822-8258

Student Investigators

Lior Berry	(604) 307-2479
Wesley Coelho	(604) 739-2052
Edward McCormick	(604) 228-0909
David Sprague	(604) 872-6911
Trevor Young	(604) 872-6087

Project Purpose and Procedures

This course project is designed to investigate how people interact with certain types of interactive technology. Interactive technology includes applications that run on a standard desktop or laptop computer, such as a word processor, web browser, and email, as well as applications on handheld technology, such as the datebook on the Pocket PC, and also applications on more novel platforms such a SmartBoard (electronic whiteboard) or a Diamond Touch tabletop display.

The purpose of this course project is to gather information that can help improve the design of interactive technology. You will be asked to use one or more forms of interactive technology to perform a number of tasks. We will observe you performing those tasks and analyze how the technology is used. You may be asked to complete a

number of questionnaires and we may ask to interview you to find out your impressions of the technology. You will be asked to participate in at most 3 sessions, each lasting no more than 1 hour.

Although only a course project in its current form, this project may, at a later date, be extended by one or more of the student investigators to form the basis of his/her thesis research.

Confidentiality

The identities of all people who participate will remain anonymous and will be kept confidential. Identifiable data will be stored securely in a locked metal filing cabinet or in a password protected computer account. All data from individual participants will be coded so that their anonymity will be protected in any project reports and presentations that result from this work.

Remuneration/Compensation

We are very grateful for your participation. However, you will not receive compensation of any kind for participating in this project.

Contact Information About the Project

If you have any questions or require further information about the project you may contact **David Sprague (604) 872-6911 or dsprague@cs.ubc.ca**

Contact for information about the rights of research subjects

If you have any concerns about your treatment or rights as a research subject, you may contact the Research Subject Information Line in the UBC Office of Research Services at 604-822-8598.

Consent

We intend for your participation in this project to be pleasant and stress-free. Your participation is entirely voluntary and you may refuse to participate or withdraw from the study at any time.

Your signature below indicates that you have received a copy of this consent form for your own records.

Your signature indicates that you consent to participate in this project. You do not waive any legal rights by signing this consent form.

I, _____, agree to participate in the project as outlined above. My participation in this project is voluntary and I understand that I may withdraw at any time.

Participant's Signature

Date

Student Investigator's Signature

Date

Sample Form Three



THE UNIVERSITY OF BRITISH COLUMBIA

Department of Computer Science
2366 Main Mall
Vancouver, B.C., V6T 1Z4

September 1, 2003

Sample Questionnaire Consent Form

Human-Computer Interaction Course Projects (CPSC 444/544)

Principal Investigator

Dr. Joanna McGenere, Assistant Professor, Department of Computer Science, University of British Columbia (604) 827-5201

Co-Investigator

Dr. Karon Maclean, Assistant Professor, Department of Computer Science, University of British Columbia (604) 822-8169

Student Investigators

Lior Berry	(604) 307-2479
Wesley Coelho	(604) 739-2052
Edward McCormick	(604) 228-0909
David Sprague	(604) 872-6911
Trevor Young	(604) 872-6087

Project Purpose and Procedures

This course project is designed to investigate how people interact with certain types of interactive technology. Interactive technology includes applications that run on a standard desktop or laptop computer, such as a word processor, web browser, and email, as well as applications on handheld technology, such as the datebook on the Pocket PC, and also applications on more novel platforms such as a SmartBoard (electronic whiteboard) or a Diamond Touch tabletop display.

The purpose of this course project is to gather information that can help improve the design of interactive technology. You are being asked to complete a questionnaire to assist us in that regard. We expect it will take you approximately 20 minutes to complete the questionnaire.

Although only a course project in its current form, this project may, at a later date, be extended by one or more of the student investigators to form the basis of his/her thesis research.

Confidentiality

The identities of all people who participate will remain anonymous and will be kept confidential. Identifiable data will be stored securely in a locked metal filing cabinet or in a password protected computer account. All data from individual participants will be coded so that their anonymity will be protected in any reports, research papers, thesis documents, and presentations that result from this work.

Remuneration/Compensation

We are very grateful for your participation. However, you will not receive compensation of any kind for participating in this project.

Contact Information About the Project

If you have any questions or require further information about the project you may contact **David Sprague (604) 872-6911**.

Contact for information about the rights of research subjects

If you have any concerns about your treatment or rights as a research subject, you may contact the Research Subject Information Line in the UBC Office of Research Services at 604-822-8598.

Consent

We intend for your participation in this project to be pleasant and stress-free. Your participation is entirely voluntary and you may refuse to participate or withdraw from the study at any time.

Your consent to participate in this project is assumed once you have completed the questionnaire.

Appendix I: Completed Questionnaires