PENUMBRA:

ENHANCING ECLIPSE FOR INTRODUCTORY PROGRAMMING

A Thesis

Submitted to the Faculty

of

Purdue University

by

Frank Mueller

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

May 2004

Für meine Eltern

ACKNOWLEDGMENTS

I would like to thank my advisor Antony Hosking for giving me the chance to work on this project. He supported me throughout the development of this work and provided me with valuable input. Furthermore, I would like to thank Sarah Caruthers for her ideas on Penumbra. She also helped testing Penumbra and provided me with editorial help on this thesis. My work on Penumbra was sponsored by an Eclipse Innovation Grant provided by IBM.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Mueller, Frank. M.S., Purdue University, May, 2004. Penumbra: Enhancing Eclipse for Introductory Programming. Major Professor: Antony L. Hosking.

In recent years, Java has become the most popular programming language for introductory programming courses; however, there has been a lack of a good Java development environment for novice computer science students. We decided to adapt Eclipse for teaching purposes by developing a plug-in called Penumbra. Through our experience teaching with standard Eclipse, as well as research on alternative environments, we determined that the goals of the plug-in should be simplicity, process integration, focus on Java concepts, easy transition to professional Eclipse, as well as specialized help documentation. Penumbra has been used by approximately 350 students in the introductory computer science course at Purdue University for the last two semesters with great success. Eclipse has proved to be an excellent platform for a teaching environment, and we will continue to develop Penumbra to expand and improve features.

1 INTRODUCTION

1.1 The problem

In recent years, Java has become the most popular programming language to be used in introductory computer science classes. While Java has proven to be effective in teaching students imperative and object-oriented concepts, there has been a lack of a good development environment for novice students. Most environments are aimed at professional programmers that are already familiar with Java or will be able to adapt to it quickly. The environments overwhelm the inexperienced programmer with details that he or she is not familiar with, and as such distract and discourage the student from learning.

1.2 Current state

Our department had been using Microsoft's Visual Studio for several years in our introductory programming class, CS180. The use, however, was discontinued because of problems with Visual Studio. The administration decided to return to a combination of Emacs, the Java Software Developer Kit (SDK), and several other tools. While this combination is fast and stable, it has several shortcomings for inexperienced users. The focus is shifted to learning the environment (e.g., Unix and Emacs) rather than the syntax of the programming language. Knowing that the current solution was insufficient, the department continued the search for an alternative environment.

1.3 Why Eclipse?

Eclipse was originally developed by "IBM, Object Technology International (OTI), and eight other companies" [1]. It is a Java development environment, as well as a tool integration platform that was made available as open source. To foster the development of Eclipse, as well as plug-ins for Eclipse, an open source community was established. "The philosophy behind the Java development tools in Eclipse is to make you as productive as possible by automating mundane and time-consuming operations and by providing tools to help you generate, edit, and navigate your Java code." [1] The features are well developed and easily rival the best environments available. However, Eclipse is not only an IDE. The tool integration platform's purpose is to help bring together all available developer tools. To facilitate this goal, Eclipse contains a sophisticated plug-in framework that allows other tools to seamlessly integrate into the Eclipse environment. Furthermore, Eclipse runs on most operating systems currently available, and it can be obtained at no cost.

In 2002, our department became interested in using Eclipse in our introductory programming class. Subsequently, Professors Hosking and Palsberg applied for an Eclipse Innovation Grant by IBM to adapt Eclipse for our purposes. After they received the grant, I was appointed as a research assistant in January 2003 to design and develop a plug-in for introductory programming.

1.4 Penumbra

The initial work consisted of evaluating the existing Eclipse environment in a teaching environment, as well as collecting requirements for the desired plug-in. To gather requirements, I also researched alternative solutions. There are numerous programming environments available that are currently being used in introductory programming classes. Some of them have been designed specifically to be used by beginning programmers, whereas most are designed for experienced users. I analyzed some of the most prominent environments to determine their strengths and weaknesses. From this information, as well as my experience in teaching, I formulated the requirements of an effective environment for teaching purposes. The plug-in was to simplify the appearance and functionality of Eclipse. Furthermore, it should integrate processes essential for checkout and submission of projects and provide features to shift the focus to learning Java. Yet, the interface of the plug-in should be similar to Eclipse so that it would be simple for the students to switch to the full-blown version of Eclipse. A help tutorial was also to be added to provided the students with step-by-step instructions for all essential features.

After presenting the goals of the plug-in at International Conference on Software Engineering in Portland, Oregon in May of 2003, I started on the development of the plug-in during the summer. The first production version of Penumbra was deployed in August of 2003 to be used in the upcoming Fall semester. New features have been added to the plug-in since then, and others have been refined and improved. Penumbra has been used by approximately 350 students in CS180 in the last two semesters.

1.5 Evaluation

Evaluation of the plug-in is subjective because we cannot perform experiments to accurately measure the success of Penumbra. Nevertheless, I have evaluated the success of Penumbra by drawing on my teaching experience in the use of standard Eclipse versus Penumbra. Moreover, I conducted two surveys in our introductory programming class in which the students assessed Eclipse and the plug-in. This will show the reader that Penumbra has achieved its goals.

1.6 Overview

The thesis is organized as follows. In Chapter 2, I discuss development environments that have been used and are currently in use to teach programming in Java. Chapter 3 describes the features that our environment, Penumbra, should include.

In Chapter 4, I explain the implementation details of the Penumbra plug-in. Chapter 5 evaluates the success of our environment, and in Chapter 6, I outline future plans of this project. Finally, Chapter 7 summarizes our conclusions.

2 BACKGROUND

2.1 The “right” environment for an introductory Java programming class

Several concerns arise about the use of an integrated development environment (IDE) for an introductory Java programming class. One major concern is the complexity of professional IDEs. As Kölling points out in [2] some educators argue that a complicated development environment will increase the number of problems beginning students face. They argue that it suffices if students use a simple text editor in combination with the Java SDK to complete their programming assignments. This group of educators is afraid that the focus will shift from learning how to program properly to learning how to use the IDE. Yet, another argument against the use of IDEs is the concern that students will not learn the details associated with a programming language, because they receive too much help from the environment. Another group of educators agrees that the use of an IDE makes sense. However, they argue that students should be handed a professional environment without any modification. In their minds students will have to learn how to use a full-blown IDE at some point in their careers. Hence, they believe that students would benefit from learning how to use a professional IDE, while they are learning how to program. Though there are valid points to these arguments, they fail to address all benefits and disadvantages of the use of an IDE. I have found that the best solution is indeed a simplified programming environment.

The use of a simple text editor such as Emacs, vi, or Notepad may initially seem to be the best choice. These editors offer a limited set of functionality that the students can quickly learn. However, the use of such editors does not come without any drawbacks. First and foremost these editors do not provide any graphical support that will help the students understand the object-oriented concepts of Java, as was also noted by Kölling in [2]. Students have immense trouble understanding the relationships between classes and objects. Graphical aid should be provided to encourage students to experiment with these concepts. Moreover, these editors provide no or only limited support for code navigation to allow the students to browse through code quickly. In addition, the choice of editor being used usually depends on the operating system. This may cause the students to use different editors depending on their location (home, dorm room, lab), which also adds problems because the students will need to learn these editors and the instructors will have to be able to help the students with them. Finally, due to the limited functionality of these editors the students will need to employ additional tools in order to complete their programming assignments. For example, students will need to be proficient in the use of SCP or FTP. To transfer their files, they will have to use a tool to submit their projects, and another tool to retrieve the initial assignments. This collection of tools easily becomes so overwhelming to students that the use of one IDE that integrates all this functionality will be the better solution.

While it is true that the use of a development environment provides immense help to students with respect to programming syntax we do not believe that students will learn less. In fact, students might learn more, because they will find it much easier to solve syntactical problems in their programs without having to consult textbooks or instructors. This will motivate them to complete their programming assignments without getting stuck on minute details. Furthermore, future IDE's will provide increased support to detect and resolve syntax errors, such that it makes only sense that students become familiar with such tools. Students should feel motivated to explore as many concepts as possible without getting frustrated about details.

Even though the use of a development environment makes sense, it is certainly true that the wealth of features can simply be overwhelming to new students. Consequently, some educators have shied away from using a professional IDE and decided to implement specialized introductory IDEs, such as BlueJ [2] or DrJava [3]. However, these specialized IDEs are no longer useful after the students completed the introductory class. Once again, students are left to learn how to use a new, more complex environment. Hence, our goal is to simplify a full-blown environment to make it more suitable to be used by beginning students. This would allow students to learn how to program in a simple environment, while the switch-over to the full-blown version of the same IDE would be relatively pain free.

The Eclipse IDE, originally developed by IBM and then released to the Open Source community, is one of the most prominent environments for programming in Java. Its success is not only based on the wide range of features, but also on the plug-in structure, which allows anybody to write software that can interface with Eclipse. A plug-in can simply add functionality, but it can also be used to customize the "Look & Feel" of Eclipse. In 2002, the Purdue Computer Science Department became interested in Eclipse sparked by the participation of Jens Palsberg at the Eclipse Birds of a Feather at OOPSLA 2002. Thus, in Fall of 2002, Jens Palsberg and Antony Hosking applied for and received an Eclipse Innovation Grant from IBM to develop a Java development interface tailored to our introductory programming class.

2.2 Experiences with plain Eclipse

With the beginning of the Spring 2003 semester, we started a special section of our introductory programming class, which consisted of students that volunteered to use Eclipse for their projects and labs. We had the students use the full Eclipse IDE, because we wanted to see what problems students would encounter during daily use. I was the teaching assistant assigned to this special section, teaching all labs and recitations. I observed the students using Eclipse in lab, and I helped them through projects during help hours and via email in order to observe them while using Eclipse. At the end of the semester I also conducted a survey to ask to students about their experience with Eclipse. The overall response by the students was very positive. Students thoroughly enjoyed the code assist features that would help to more quickly fix bugs in their programs, as well as some of the simple refactoring features to format code and organize imports. However, the students also experienced numerous problems when using Eclipse, especially at the beginning of the semester. Some trivial tasks such as setting up projects and running them are complex procedures in Eclipse that require several steps. Students needed detailed instruction in order to complete these tasks. This is due to the fact that some of the abstract concepts, such as projects, are not part of Java, but are rather introduced by Eclipse. While these abstractions are essential for professional programmers they distract the beginning students, so I determined there was a definite need to simplify many steps to keep the students focused on their programming assignments. It is not always possible though to completely eliminate some aspects, so I furthermore argued that we needed to provide the students with a detailed help tutorial to walk them through fundamental aspects of Eclipse. While Eclipse provides some great features to help the programmer visualize program structures, they are mostly hidden, difficult to configure, and a generally not designed for use by novice programmers. Hence, another objective of our plug-in was to provide the students with an interface that allowed them to intuitively interact with object and classes.

2.3 Non-Eclipse solutions

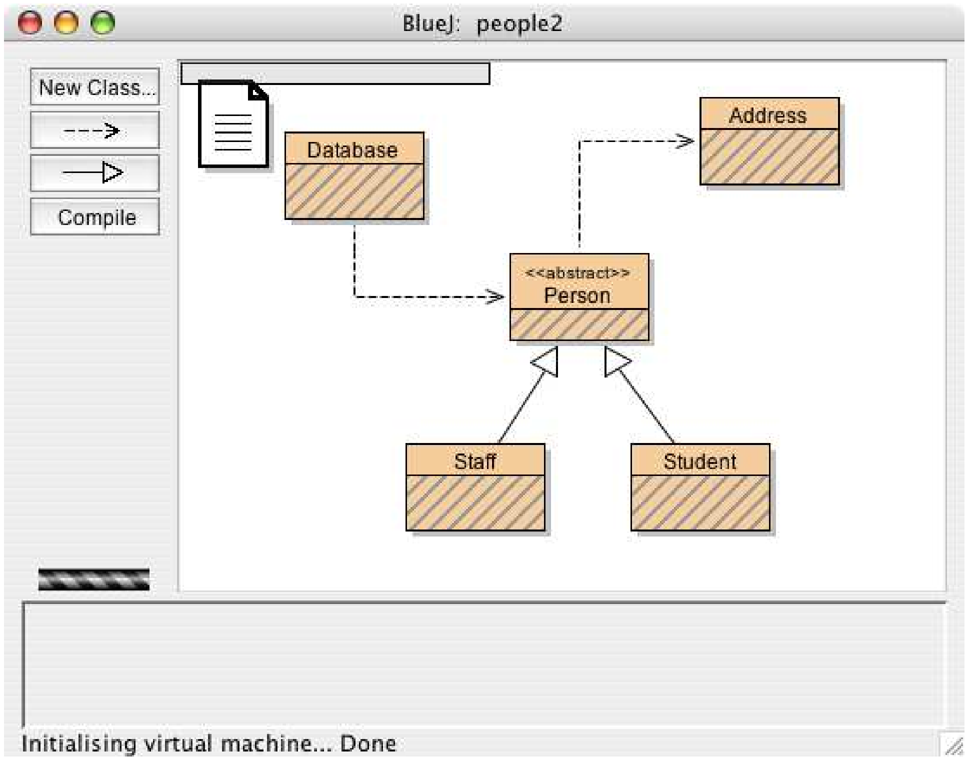
While we believe that Eclipse is one of the best platforms to provide a simplified IDE for introductory Java programming classes, there are certainly other solutions available. In this section, I will discuss solutions that have been formerly employeed at Purdue University or are currently in use at other institutions. The IDEs presented in this section do not rely on Eclipse. I will discuss alternative Eclipse solutions in the next section.

2.3.1 Visual Studio

Visual Studio by Microsoft Corporation is a well known environment. At the time we decided to teach Java in our introductory programming classes, it was in fact one of the most prominent IDEs. We decided to use Visual Studio because we believed that IDEs would help students with their programming assignment. The environment provided excellent code assist features, some of which were later adopted by Eclipse. However, the use of Visual Studio was not without problems. It turned out that students were quickly overwhelmed by the wealth of powerful features. The environment allowed the addition of certain features, but it did not allow modifications to such an extent as Eclipse. Furthermore, Visual Studio is an expensive IDE, which only runs on Windows. The dependence on Windows was perceived as a problem by our department, which traditionally teaches most courses in Unix environments. Eventually, Microsoft discontinued the Java support in Visual Studio which became increasingly problematic with the surfacing of newer versions of Java. Instead Microsoft was pushing for C#, which it just recently developed. Consequently, the use of Visual studio was dropped after two years and the department opted for the use of a simple editor that the students used to write their code. They then compiled and ran their programs from the Unix command prompt. The need for a teaching IDE remained and so the search continued.

2.3.2 BlueJ

BlueJ is probably the most widely used and best known environment specifically designed for the purpose of teaching Java programming classes. It was originally developed at Monash University in Melbourne, and is now maintained by a joint research group, with members in Australia, Europe and North America. As described by Kölling in [2] [4] [5] it was developed for the sole purpose of being used in a teaching environment. Like Eclipse, BlueJ was written in Java and so it runs on all platforms supporting a Java virtual machine. There are supported versions for



Initialising virtual machine... Done

Figure 2.1. BlueJ: An introductory IDE

Windows, Linux, and Macintosh OS X available on the BlueJ website. Furthermore, it can be obtained at no cost. Since 1999 BlueJ has been used by more than 300 colleges and universities worldwide.

What are the reasons for the success of BlueJ? Surely, the fact that BlueJ runs on most systems and is available for free, makes it a good candidate for an introductory environment, but there are more important reasons. From the start, the BlueJ project had two goals . It wanted to provide an environment that "encourages the students to think in terms of objects and classes" and "does not distract from the task of learning to design and implement a program" [2]. In order to accomplish the first goal, BlueJ represents all classes in a project as objects in a flat hierarchy with arrows drawn between these objects to symbolize "inheritance" and "uses" relationships as shown in Figure 2.1. Furthermore, BlueJ allows the user to test classes by instantiating an object of a class and running its member methods. Testing can be done using drag and drop without having to write additional testing code. Furthermore, the environment comes with a simple editor that supports syntax highlighting, as well as a simple debugger. The entire IDE is only about 3.6 Megabytes large.

While BlueJ has been successful in providing a simple teaching environment, it has shortcomings that prevent it from being the universal environment for introductory Java classes. The editor itself is no more powerful than editors such as Emacs. It provides no code assist features, and it does not provide any help in resolving syntax errors. Designing and implementing such code assist features is a difficult task that requires a substantial number of programmers. Usually these powerful features are only included in professional IDEs. Students will not be able to use BlueJ for very long after they complete their introductory class, because they will want to use a more powerful IDE to more quickly and efficiently solve the programming assignments in their upcoming classes. Eventually, students will have to transition to another IDE, which will leave them in a similar position they found themselves in at the beginning of their college careers. Transition should be made as easy as possible.

2.3.3 DrJava

DrJava is another environment specifically designed for the purpose of teaching introductory Java classes. It was developed at Rice University under the supervision of Robert Cartwright. Like BlueJ, it is written in Java and as such should be runnable on any system with a Java virtual machine. Another advantage is that it is also available for free. While DrJava is being used by schools around the world, it has not had the success of BlueJ. DrJava's emphasis has been placed on inter-activeness rather than visualization. The environment does not provide visual support to allow students to interact with classes and objects. The interface of DrJava consists of three components: an interactions pane, an editor [3] [6], and a simple debugger. The editor and the debugger do not provide any exceptional functionality, but are kept simple. However, the editor provides the ability to explore libraries. This features is essential for working in more complex projects at the end of the first semester.

The most interesting feature of DrJava is the interactions pane, which works similar to an interpreter. It utilizes a "read-eval-print-loop" (REPL) [7] that students can use to interactively experiment with Java expressions and statements. For example, students may instantiate an object of a class and call its member methods. The result of the method calls will be outputted on the screen. To implement the REPL, DrJava relies on DynamicJava [8], which is freely available. The interactions pane can also be used to more quickly debug programs, because specific methods can be tested extensively without having to run the entire program.

The problems of DrJava are similar to those associated with BlueJ. The editor is very simple and provides no code assist features. Students will need to learn a different IDE because DrJava is not designed to manager larger projects that students will have to tackle in later classes. In order to solve this problem, the DrJava team has also implemented a plug-in for Eclipse. This will be discussed in the next section.

2.4 Eclipse plug-ins

Since IBM released Eclipse to the Open Source community, it has been pushing for the development of new plug-ins. In this section, I will discuss two of the most interesting plug-ins for educational purposes.

2.4.1 GILD

GILD stands for "Groupware enabled Integrated Learning and Development" [9]. It is an Eclipse plug-in developed at the University of Victoria with the support of IBM Canada. Much like Penumbra, it was designed specifically to be used with Eclipse, and as such makes extensive use of the Eclipse framework. Superficially, GILD seems to be very similar to Penumbra, but there are some deep seeded differences, which are based on the different philosophies. While Penumbra wants the students to use Eclipse's powerful features in order to complete their assignments, GILD is trying to revert Eclipse to a very simple Java editor and debugger. Consequently, code assist and refactoring features have been made unavailable to the students in GILD. Furthermore, the GILD team is putting more emphasis in providing a perspective for instructors to incorporate a content management system that allows for a tight integration between lecture materials and programming exercises [10]. Students can study their materials using a simple web browser integrated into Eclipse. GILD allows the instructors to make references between course materials and projects. This functionality allows the student to see how a subject matter relates to coding problems that they can run and experiment with. Furthermore, the GILD team is currently developing instant message capabilities for pair programming assignments.

The problems of GILD are its lack of integration of processes for distributing and submitting programming assignments. Furthermore, there has been no effort made to provide the students with tools that focus on object-oriented concepts. To alleviate this problem GILD has implemented DrJava's interaction pane, which is a poorly visualized solution.

2.4.2 DrJava

As pointed out in the previous section, the DrJava development team was aware that its IDE is of little use to advanced students that need a powerful IDE to work on complex projects. Hence, they also became interested in using Eclipse as a basis for their introductory IDE [11]. However, the resulting solution is not well integrated into Eclipse. The DrJava interactions pane and the core runtime environment were simply ported to be accessible from inside Eclipse. The plug-in makes very little use of the Eclipse framework. No effort was made to simplify the interface or improve the visualization of the interactions pane. The pane itself it very useful and allows students to quickly test objects and methods, but it should have been more tightly integrated to show the students the objects they are working with. Furthermore, the plug-in does not contain any additional features that were not contained in the original DrJava IDE.

3 DEVELOPMENT GOALS

After assessing our in-class experiences, conducting the survey at the end of the semester, and doing extensive background research, I defined the issues that I wanted to address in the Penumbra plug-in. Each goal is described in detail in this section, and its implementation is discussed in the next chapter.

3.1 Simplicity

Like most IDEs, Eclipse has an overwhelming arsenal of tools to assist professional programmers. In order to make these tools available to programmers, Eclipse relies on a collection of menus, tool bars, and views. With the addition of each menu or toolbar, the appearance (perspective) of Eclipse grows more complex. This increasing complexity makes it more difficult for the students to locate the features they consider useful. Hence, the most important aspect of Penumbra was to design a simplified perspective. It should only consist of four views - a package explorer, an editor, an object-oriented hierarchy view, and a command line view that allows for input and output. The Penumbra package explorer should be very similar to the default Java package explorer, but its default filter should be adjusted to display only the student.s projects and no reference to libraries. Furthermore, I wanted to remove unnecessary menus and options associated with the package explorer to provide a clean look. The editor and the command line view were not to be changed, since the default had proven to be very effective. The object-oriented hierarchy view will be discussed in detail in one of the following subsections.

As pointed out earlier, not only is the appearance of Eclipse complex, but also the way most of its features work are complicated. For example, the process of setting up a Java project containing a single class, requires a series of features to be used.

While students can be walked through this process during lab hours, the students will have to conquer such challenges when they try to use Eclipse by themselves. In order to alleviate these problems, I decided to significantly simplify the steps required to set up projects and run them. Students should be able to complete these tasks with one or two mouse clicks, allowing them to focus on experimenting with the class and not setting it up.

3.2 Process integration

Our experience in past semesters had shown that students needed to utilize a suite of tools in order to obtain project specifications and skeleton code, set up the project, and finally submit the project for grading. Tools employed would include a web browser, file management tools, and a submission tool. This meant students had to be comfortable using all these tools, even though they did not relate to the actual course content. Since, most students were not familiar with these tools, time had to be used in lab hours and recitations to explain them in detail. Furthermore, using these tools made working from home awkward for students. For instance, if students wanted to work from their Windows computers, they had to either use SSH to be constantly connected (which is often times not possible on dial-up connections) to a lab computer, or they had to transfer the files back and forth using SCP, just to be able to submit the latest version for grading.

However, Eclipse already contains numerous features for file and project management. Therefore, our goal was to extend these features so that students could easily obtain and submit projects, allowing quick assessment of the assignments. Furthermore, students should be able to do so independently of their current location. Ideally, students should be able to submit their projects from home just as easily as from the labs on campus.

3.3 Focus on learning Java

As was discussed earlier, I wanted to shift the focus away from teaching the students how to use several tools to complete their assignments to learning Java itself. One aspect, is to simplify Eclipse, but another important aspect pointed out by Kø"lling is to provide the student with an object-oriented view. Beginning students have immense trouble with the object-oriented concepts of Java. Thus, I needed to provide the students with a view that would help them visualize and interact with these concepts. The view.s purpose is to show how classes and interfaces are related. Furthermore, students should be able to modify the hierarchy using simple drag and drop motions. Students should feel encouraged to change the hierarchy and see what changes this would mean for the underlying source code. While the idea of the view would be very similar to the one used by BlueJ, its implementation would more closely resemble other views shown in Eclipse.

3.4 Easy transition to full-blown Eclipse

Our perspective for Eclipse was to be simple and intuitive for beginning students. However, I was not interested in building a plug-in that would remove any resemblance to the complete version of Eclipse. I wanted to preserve the key elements of the standard Java perspective to allow the students to easily switch to the full-blown Java perspective when ready. Once students feel comfortable with the subset of functionality provided to them, they will be able to move on to the complete perspective. Some advanced students might feel inclined to switch during the course, while other students might use Penumbra for the duration of the entire introductory class. Either way, students should feel comfortable enough to work on their projects after the switch, which will likely be necessary in one of their next programming classes.

3.5 Help tutorial

Eclipse provides a variety of help documents for its environment. However, this documentation is inadequate for inexperienced programmers. Much like man pages on Unix systems, the focus is on providing detailed help on specific features. However, novice programmers often do not know what it is they are looking for. They need help documentation that will walk them through the most basic features of Eclipse and Penumbra. Our goal was to create help documents that would discuss issues, such as setting up projects, submitting projects, and using the debugger included in Eclipse. Although it is my goal to simplify the essential functionalities in Penumbra as much as possible, there will always be a need for detailed documentation for the students. Furthermore, my help was not only designed to point out features that are not necessarily essential to understand, but are extremely useful, such as some of the refactoring features. Students are much more likely to use code-assist features, such as Organize Imports, if we point it out to them and have them experience the benefits. The overall goal of the help documentation was of course to provide detailed help, even in the event that a lab instructor is not at hand, and a student needs to solve the problem.

4 IMPLEMENTATION

All implementation details described are with respect to Eclipse version 2.1.x. A new version 3.0 of Eclipse is still in its developmental phase. Porting Penumbra to this new version will be addressed in a later chapter.

4.1 Plug-in structure

The fully qualified name of the plug-in is edu.purdue.penumbra. It consists of 60 classes organized in 7 packages. The overall structure can be seen in the Figure 4.1. The two most important files in the plug-in are plugin.xml and PenumbraPlugin.java, contained in the main package. Plugin.xml defines all properties of the plug-in. These properties include a listing of the plug-ins required for Penumbra to run properly, as well as detailed declarations of all extensions that the Penumbra plug-in will add to Eclipse. Consider the extract of code in Figure 4.2. It specifies that two views should be added: the Package Explorer and the hierarchy view. Furthermore, it defines the classes that need to be instantiated to display the views, as well as a name and a logo that can be used to refer to them in menus and toolbars.

In a similar manner, all other extensions the Penumbra plug-in adds to Eclipse are declared in plugin.xml. The class PenumbraPlugin is used as an interface to make all extensions accessible to Eclipse. It contains few lines of source code, but is crucial for the interaction between Penumbra and Eclipse. Furthermore, SavitchIn [13], a collection of methods to simplify user input, is packaged with Penumbra. SavitchIn has been used in CS180 for several years and is crucial for the first projects in CS180. There are two reasons that led to the decision to include SavitchIn with Penumbra. All students are required to use this class to do their projects. Hence, adding it to Penumbra eliminates an extra installation step for users at home. Moreover, placing

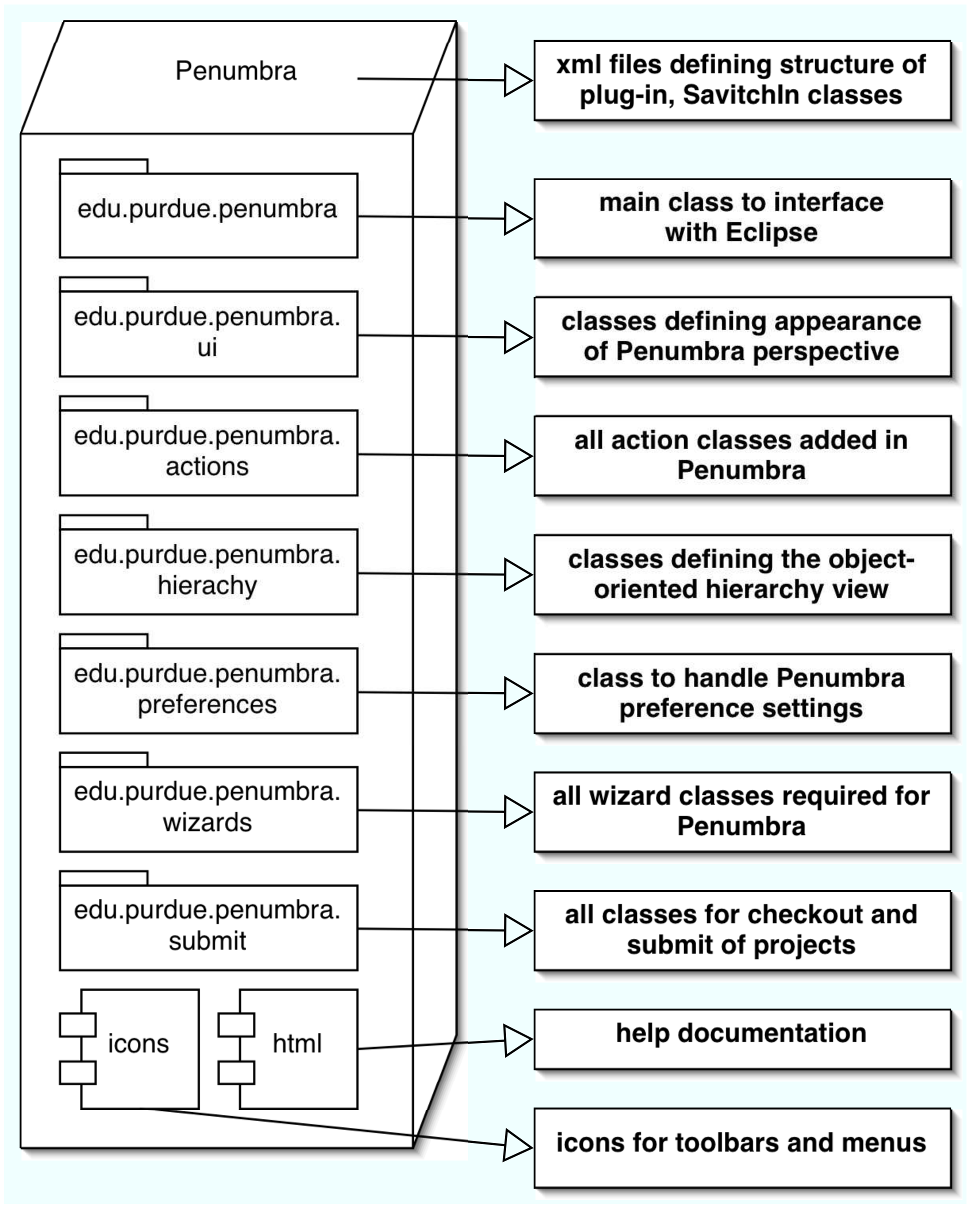

Penumbra
xml files defining structure of plug-in, SavitchIn classes
edu.purdue.penumbra
main class to interface with Eclipse
edu.purdue.penumbra.ui
classes defining appearance of Penumbra perspective
edu.purdue.penumbra.actions
all action classes added in Penumbra
edu.purdue.penumbra.hierachy
classes defining the object-oriented hierarchy view
edu.purdue.penumbra.preferences
class to handle Penumbra preference settings
edu.purdue.penumbra.wizards
all wizard classes required for Penumbra
edu.purdue.penumbra.submit
all classes for checkout and submit of projects
icons
html
help documentation
icons for toolbars and menus

Figure 4.1. Package structure of Penumbra

```
<extension
         point="org.eclipse.ui.views">
      <view
            name="Penumbra Package Explorer"
            icon="icons/penumbraLogo.jpg"
            class="edu.purdue.penumbra.ui.PenumbraPackageExplorer"
            id="edu.purdue.penumbra.ui.PenumbraPackageExplorer">
      </view>
      <view
            name="Project Hierarchy"
            icon="icons/penumbraHierarchy.gif"
            class="edu.purdue.penumbra.ui.hierarchy.PenumbraHierarchyView"
            id="edu.purdue.penumbra.ui.TypeHierarchy">
      </view>
   </extension>
```

Figure 4.2. Sample of plugin.xml

SavitchIn in the plug-in specifies exactly where it is to be located on the student's computer. It can then easily be added to the CLASSPATH, without the students having to modify the CLASSPATH themselves in order to use SavitchIn.

In the following sections, I will explain implementations of all Penumbra features. The nature of the plug-in structure makes it inconvenient to explain package by package what each class is used for, because classes are organized by their type, rather than the feature they belong to. Therefore, the description will be according to features as they have been specified by the goals in the previous chapter.

4.2 Penumbra perspective

A perspective in Eclipse, defines the menus, toolbars, and views the user sees. As such the Penumbra perspective is undoubtedly one of the most essential parts of the plug-in. The Penumbra perspective is based on the Java perspective, which is the default programming perspective in Eclipse. Instead of designing a perspective from scratch, I adapted the Java perspective found in org.eclipse.jdt.internal.ui.

The core of the Penumbra perspective is a class called PenumbraPerspectiveFactory. When instantiated a call will be made to the method createInitialLayout. This method defines where on the screen certain views will be opened. In the case of Penumbra, it will open the Penumbra Package Explorer on the top left side, and the Penumbra hierarchy view on the bottom left side. Furthermore, the console will be placed on the bottom. The Java editor will not be created at this time. It will be instantiated once a Java source file is opened. Notice that the perspective factory does not allow us to modify the menus and toolbars that are visible to the users. Menus and toolbars are not directly connected to a perspective, but rather belong to a particular view. For example, if the Java editor is opened for the first time, it will create menus that will allow the user to build and run the code. The Eclipse plug-in structure was designed with the idea that users would only want to add functionality, but not remove any of it. This design flaw has been addressed in the upcoming version of Eclipse. However, I had to find a somewhat cumbersome solution to work around this design flaw in version 2.1.x of Eclipse. As pointed out previously, when a view is opened for the first time, it will create menus and toolbars that are associated with this views. These menus and toolbars are called ActionSets in Eclipse. In order to add an AcionSet, the view will have to register it with the current perspective. Unfortunately, it is impossible to keep the views from adding their ActionSets without having to modify them, which would have lead to an unnecessary duplication of many classes.

My alternative solution was to remove these ActionSets right after they had been added. To accomplish this, I designed a class called PenumbraSwitchPerspectiveAction. When instantiated, this class will switch to the Penumbra perspective. It will then scan the array of ActionSets and remove all unwanted menus and toolbars. When Penumbra is installed it will set Penumbra to be the default perspective and make a call to PenumbraSwitchPerspectiveAction. All subsequent openings of the views will not add the lost ActionSets to the Penumbra perspective, and so the user will not notice the "trick". Besides removing numerous menus and toolbars, a menu called Penumbra was added, along with a toolbar with shortcuts to these menu functions. The Penumbra menu contains all functionality that was added by Penumbra, or functionality that was otherwise hidden in submenus in the Java Perspective. For example, there is an action to format the source code. Students might not use this action because it is placed in a menu they would not usually open. Consequently, I decided to move this item, as well as other useful functionality, into a common menu in order to encourage students to experiment with these options.

Next, I simplified the Package Explorer, which has all functionality required, but is simply too complex for beginning programmers. It contains menus and filters that novice programmers will not touch, and so they will only add to the distraction of the overall perspective. Hence, I chose to adapt the default Package Explorer found in org.eclipse.jdt.internal.ui.packageview. I modified the view such that all of its menus were disabled, and, furthermore, I preset the filters to allow students to see only the content in their projects. For example, by default, the Java Package Explorer shows all library references, which adds confusion and makes it harder for students to understand the files they are responsible for.

Part of the simplification process was also the need to redesign the way students set up and run projects. As described earlier, it was a goal to allow the students to easily set up projects and run them, without having to go through many steps. By default, students need to set up a project, then they need to create a class, and they need to remember to include a main method. Furthermore, the CLASSPATH has to be set up to include SavitchIn [13], a collection of methods used to help students obtain user input. While these steps seem trivial to the experienced programmer, they do, in fact, cause many problems for novice programmers. I wrote a project creation wizard that merely asks the student to input a name, and the wizard then sets up a project that contains a class with the same name. The class automatically contains a main method, and the CLASSPATH for the project is set to include SavitchIn. The class PenumbraNewProjectCreationWizard makes use of the Java project and class creations wizards contained in Eclipse. It first creates a project with the CLASSPATH containing SavitchIn. It then passes the newly created project to the default wizard for class creation, which is then fed the class name and the request for a main method. The class wizard is automatically executed, and the new project will appear in the Package Explorer.

Running projects in Eclipse is by default more complicated than it is in a Unix console. This is due to the fact that Eclipse allows the programmer to run programs not just as Java Applications. For instance, projects may be executed as applets or as plug-ins in another instance of Eclipse. However, this requires the user to specify exactly the environment in which he or she would like to run the program. Many properties can and must be specified in order to run programs. In my opinion, students should only have to select a project or class containing a main method and click a single button to run it. However, simplifying the procedure this much would cause other problems. For instance, what if students want to pass command line arguments? What if they would like to run an applet? Thus, I selected a subset of cases that do occur in CS180 and made them available in a submenu (see Figure 4.3).

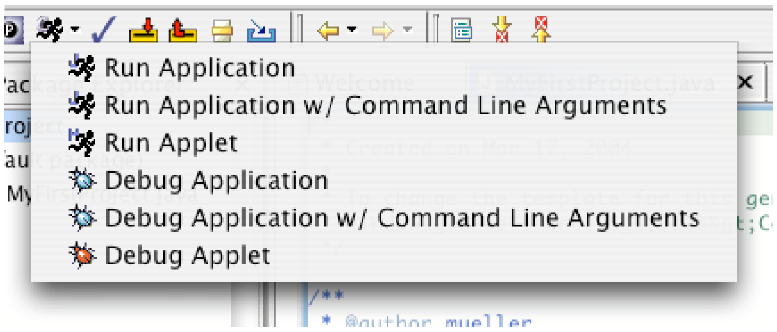
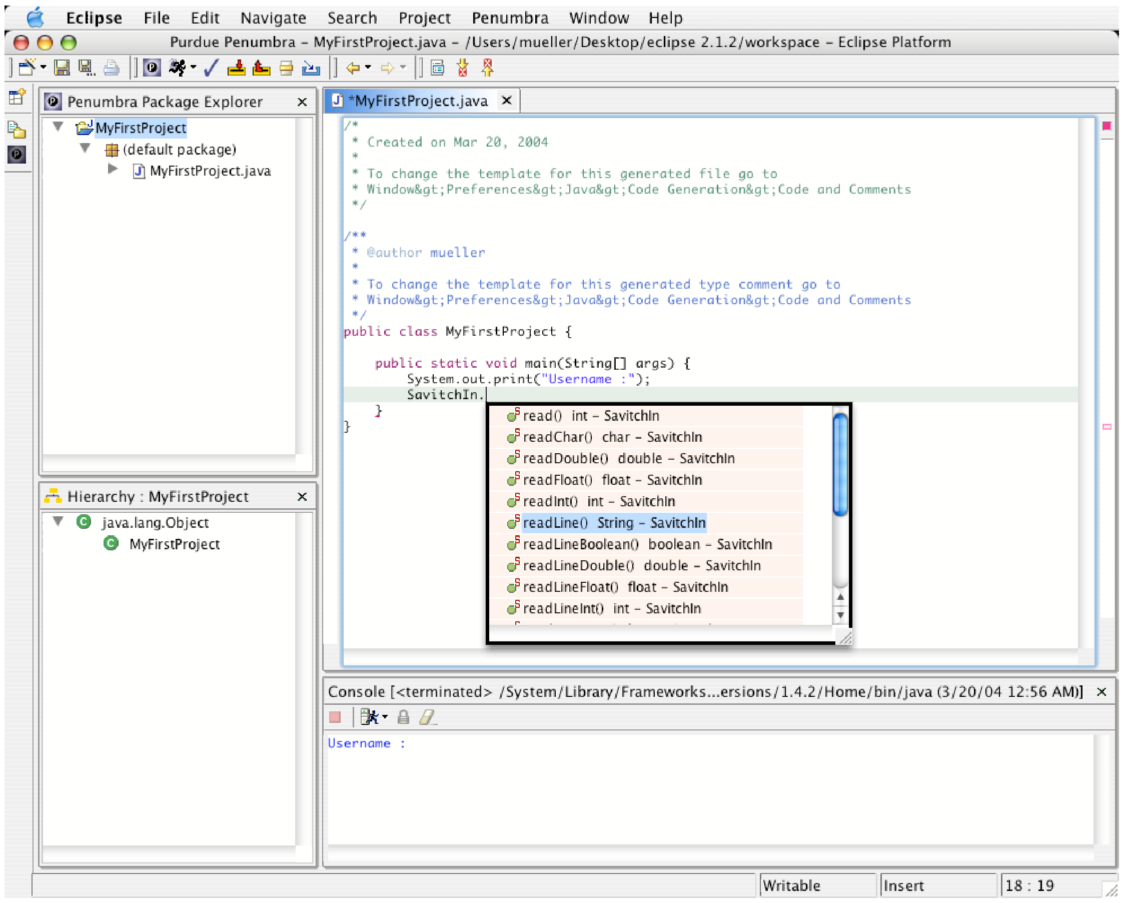

Run Application
Run Application w/ Command Line Arguments
Run Applet
Debug Application
Debug Application w/ Command Line Arguments
Debug Applet

Figure 4.3. Penumbra run menu

Whenever the user selects an element in the Penumbra Package Explorer, the Penumbra hierarchy view, or an editor window, the selection is noted by a class called PenumbraSelectionAction. If the student chooses to click any of the run buttons, it will spawn the responsible PenumbraRunAction (depending on the menu entry chosen), which will attempt to determine which class the student would like to run. If the student was working in an editor with a main method, the choice is easy, and Penumbra will simply execute this class. However, if the student last selected a project that contains multiple classes with main methods the student will be prompted to select the class he or she would like to execute. Students do not have to make any further specifications. If the student selects the application to be executed with command line arguments, the student will be prompted to enter these arguments in a pop-up window. The arguments will be saved in the Eclipse registry, so that they do not have to be re-entered at each execution.

The resulting Penumbra perspective can be seen in Figure 4.4.



Eclipse File Edit Navigate Search Project Penumbra Window Help
Purdue Penumbra – MyFirstProject.java – /Users/mueller/Desktop/eclipse 2.1.2/workspace – Eclipse Platform
Penumbra Package Explorer
MyFirstProject
(default package)
MyFirstProject.java
Hierarchy : MyFirstProject
java.lang.Object
MyFirstProject
*MyFirstProject.java

```
/*
 * Created on Mar 20, 2004
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

/**
 * @author mueller
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */
public class MyFirstProject {

    public static void main(String[] args) {
        System.out.print("Username :");
        SavitchIn.
    }
}
```

```
read()  int – SavitchIn
readChar()  char – SavitchIn
readDouble()  double – SavitchIn
readFloat()  float – SavitchIn
readInt()  int – SavitchIn
readLine()  String – SavitchIn
readLineBoolean()  boolean – SavitchIn
readLineDouble()  double – SavitchIn
readLineFloat()  float – SavitchIn
readLineInt()  int – SavitchIn
```

Console [<terminated> /System/Library/Frameworks...ersions/1.4.2/Home/bin/java (3/20/04 12:56 AM)]

```
Username :
```

Figure 4.4. Penumbra perspective

4.3 Project checkout via CVS

Another goal of Penumbra was to provide a mechanism that would allow for easy distribution of projects, containing skeleton files defining methods which must be implemented in the current project, as well as starter code and comments, to the students and their resubmission for grading. Concurrent Versions System (CVS) [14] seemed to be the ideal solution. It can be used to distribute. receive and manage software for a large group of developers. Moreover, Eclipse is already equipped with a CVS plug-in. A problem is that the plug-in provides more functionality than we need, and it is too complex to be used by novice students. A seamless solution should be to leave the user completely unaware of the underlying mechanism used to check out and submit projects, since it is not the focus of the introductory Java programming class. Furthermore, if it were necessary to change the mechanism, it should not effect the way it appears to the students.

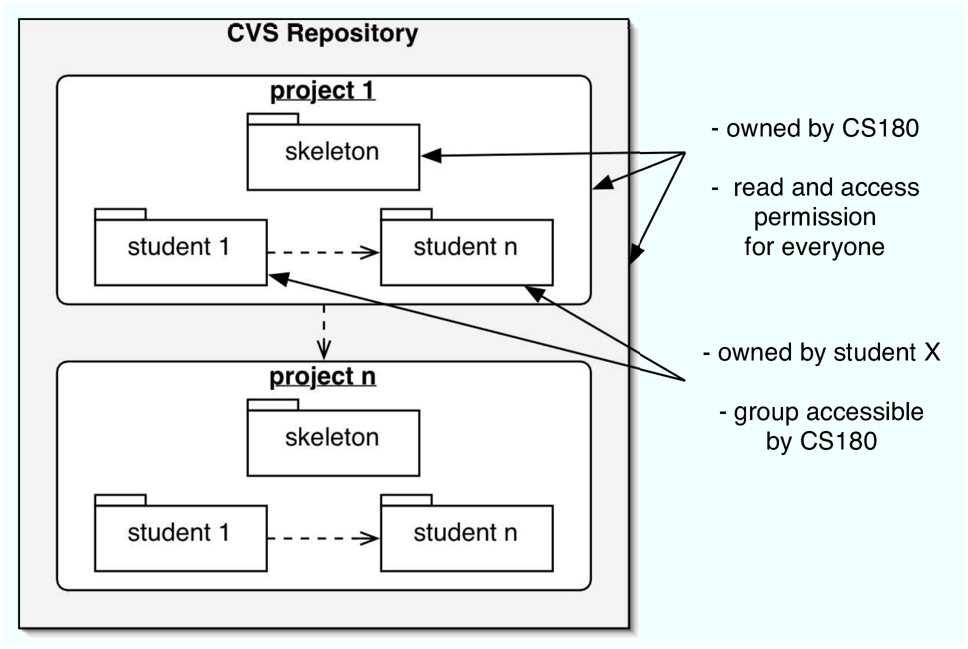
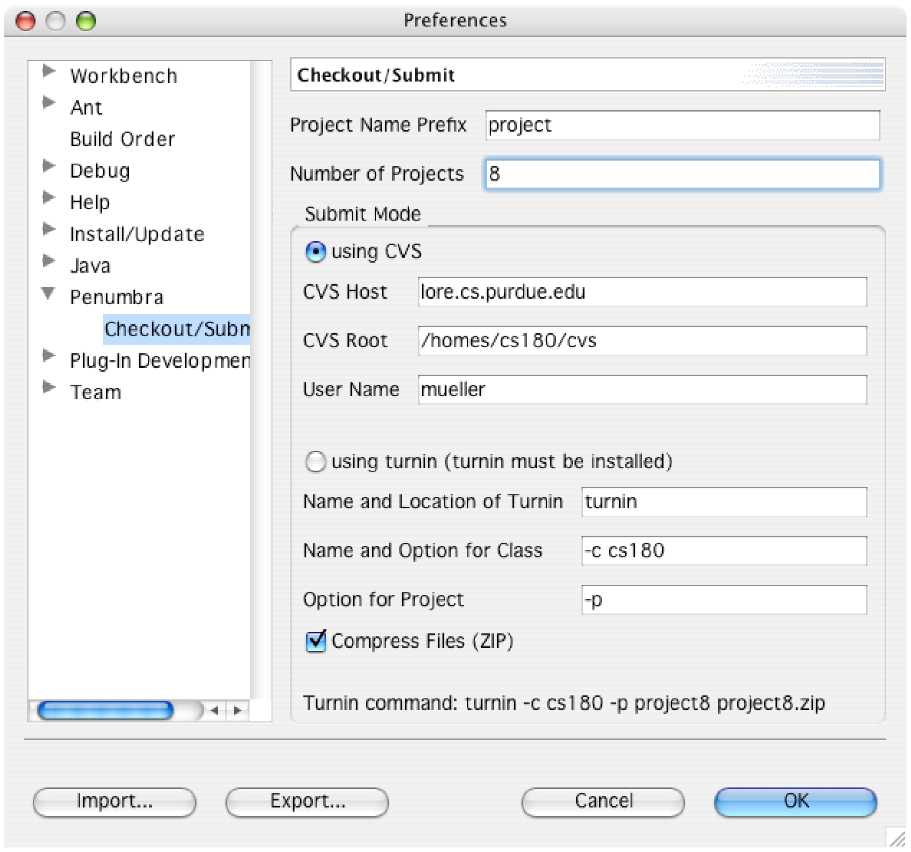


Figure 4.5. Layout of CVS repository

Unlike in Open Source projects where CVS repositories are used to foster collaboration between team members, I had to design a repository that would permit each student to retrieve and submit only their own projects. Furthermore, the respository has to be accessible to the CS180 group (instructors and teaching assistants) to be able to grade the projects. It should also be possible to turn off submission after project deadlines have passed. The layout I designed is shown in Figure 4.5. The entire repository is owned by the group CS180, but is accessible to all others. Inside the repository, I create a module for each project, which in turn contains modules for each student and an additional module called skeleton. The project modules are by default accessible to anyone, but permissions can be modified to disallow access after the deadline has passed. The modules for each student are owned by the students themselves, as to avoid anyone else gaining access to their source code. The skeleton folder holds the skeleton code that is made available to the students at the beginning of the project. I provide Unix shell scripts that populate the CVS repository automatically and that allow the instructors to easily post new projects. The script to populate the repository has to be run by root, as only he or she has the ability to create folders that belong to the students. We cannot depend on the students to set up the directories by themselves.

As pointed out earlier, the use of CVS as a means to check out and submit projects was to be included in a seamless manner. Students should not have to navigate through the CVS repository to find the project they need to check out or submit. Therefore, I developed new wizards that are based on the default Import and Export wizards, but instead of reading or writing to the file system, they connect to a CVS server. In order for Penumbra to be able to set up the connection to the CVS repository and use it without the student explicitly knowing about it, I had to gather information from the students, such as user name and the location of the CVS repository. To do so, I created a preference page. The preference page in Figure 4.6 will collect information from the user that allows Penumbra to automatically set up a connection and use it. By default, all information is set to be used in CS180, except for the user name. Once the student enters his or her user name and clicks on "OK", Penumbra will set up a new CVS location. Furthermore, the name of the location along with the other preferences will be stored in the Eclipse registry, so that it can be used subsequently without consulting the user.



Checkout/Submit

Project Name Prefix: project

Number of Projects: 8

Submit Mode

[✓ using CVS]

CVS Host: lore.cs.purdue.edu

CVS Root: /homes/cs180/cvs

User Name: mueller

using turnin (turnin must be installed)

Name and Location of Turnin: turnin

Name and Option for Class: -c cs180

Option for Project: -p

✓ Compress Files (ZIP)

Turnin command: turnin -c cs180 -p project8 project8.zip

Figure 4.6. Penumbra preference settings

In order to check out a project, the student has to select the appropriate option from the Penumbra menu or tool bar. This will open the PenumbraCheckoutWizard, which will connect to the CVS location known to Penumbra and list all project names as shown in Figure 4.7. The student now needs to select a project and click "OK" to start the checkout process. First, Penumbra checks if the project is available, which it does by looking for files in the skeleton folder of the specified project. If there are none, the project is not yet available for check out. Next, the wizard will determine if the user has previously checked out the project. This is necessary because students can submit their code and check it out at another location. Hence, students will receive the latest version of their project code unless they have not yet submitted anything. If this is the first check out the student has performed on the project, the wizard will download the code contained in the skeleton directory of the respective project. However, simply obtaining the skeleton code is not sufficient because at the next commit operation CVS would attempt to overwrite the code in the skeleton directory. After checking out the skeleton code, the checkout wizard disconnects the project from the skeleton module and reimports as the first version of the students code in his or her project module. All this happens as one uninterruptible step, in a matter of a few seconds. Due to the fact that several errors can occur during check out (network failure, CVS server failure, etc.), the wizard will verify each step in order to provide students with a detailed error message in case a problem occurs. Once the project is checked out, it will appear in the Penumbra Package Explorer and the Penumbra hierarchy view, where the student can select the files to open and start working immediately.

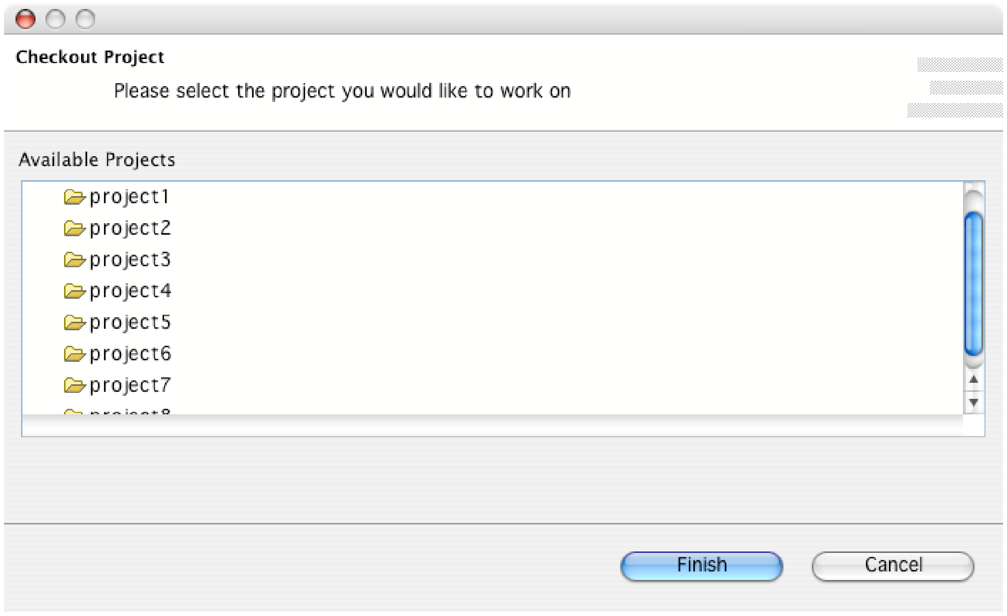


Figure 4.7. Penumbra project checkout

4.4 Project submission via CVS and turnin

I developed two mechanisms to submit projects via Penumbra. The default mechanism of choice is submission via CVS. However, there is also the option to submit projects by passing it to a command such as turnin. The submission method can be selected in the Penumbra preferences page. Submission via CVS offers greater convenience for a number of reasons. An application such as turnin is usually bound to a particular operating system. Consequently, using turnin will only work on the Unix machines at school, whereas using CVS will allow students to submit the projects from virtually anywhere independent of the operating system they are running. Secondly, using CVS will allow students to retrieve previous versions of their project source code. I have integrated a feature that allows students to easily view and retrieve previous versions of their source files. Selecting the appropriate menu option will yield a list of versions available in the CVS repository, depending on the file selected. Another double-click will download and open that version. This feature allows students to easily recover source code that they previously deleted.

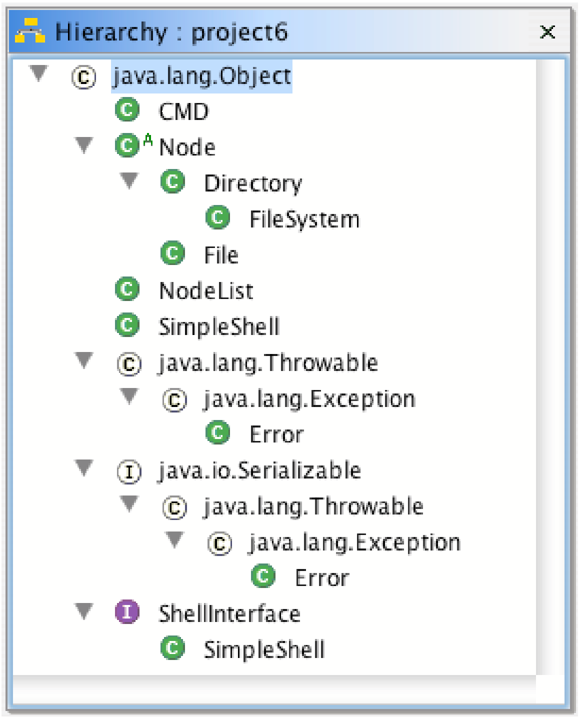
When selecting "Submit Project" from the Penumbra menu, the PenumbraExportWizard class will be instantiated. It will ask the user to select one of his or her projects. If the project has been checked out or submitted previously, Penumbra will automatically know where to submit the project. However, if the project has not been checked out or submitted before, the student will have to select from a list the name the student would like to submit the project under. The export wizard will then connect to the CVS repository in order to commit the student's project. To save storage space in the repository, only source files will be submitted. All compiled files will be filtered out by the wizard. Shortly after the first version of Eclipse was in use, I discovered that students wanted additional confirmation that their projects submitted correctly. By default, I have Penumbra display a message if the submission was or was not successful. However, students would also like to confirm the dates and times that their files were submitted. Hence, I implemented a feature that would connect to the CVS server and display the information for all files submitted for the selected projects.

Submission via turnin works very similarly from the user's perspective. It does not offer the benefits of CVS, but it is easier to administer than a CVS repository. The same wizard is invoked and asks the student to select the project to be submitted. The export wizard then filters out the sources file and submits them using the specified turnin command. If selected on the Penumbra preference page, the files will be zipped up and passed to turnin. At the end of the process, a window displays the turnin output to allow the user to judge if turnin was or was not successful. Since the output of such turnin commands differ, it is impossible to provide one solution that would allow Penumbra to judge if submission was or was not successful.

4.5 Penumbra hierarchy view

The Penumbra hierarchy view was designed to help students better understand the object-oriented concepts of the Java programming language. The goal was to provide the students with a view that would visualize the hierarchy of classes and interfaces in a project, but that would also allow for interaction. Unlike BlueJ's flat structure, I chose a tree structure to display the class hierarchy. Throughout Eclipse, views use tree structures to display the contents of projects, packages, and classes. In order to more seamlessly integrate the Penumbra hierarchy view, I adapted one of these tree structures. By default, Eclipse contains a type hierarchy viewer, which is used to display member methods of classes. This view, which can be found in org.eclipse.jdt.internal.ui.typehierarchy, provided an excellent basis for what I needed to design.

I modified the view as to display the hierarchy of one project at a time. If a different project or an element of a project is selected in the Penumbra Package Explorer, it will cause the hierarchy view to refresh its contents. The package explorer will pass the project root of the selected element to the hierarchy view. The hierarchy view in turn will recursively traverse the children of the root and build a tree of class nodes. This tree is then annotated with icons and name labels to be displayed. The fully qualified name of all objects will be displayed to show if a class or interface is part of the default package, another package within the project, or a completely different package. All classes and interfaces, even if they are from different packages, are shown in the same hierarchy, to help the students visualize class dependencies across packages. Figure 4.8 shows an example hierarchy view. Notice that due to the fact that classes can implement multiple interfaces, some classes will be shown in multiple branches. The nodes in the hierarchy are decorated with the default Eclipse icons.



```
Hierarchy : project6
▼ java.lang.Object
      CMD
  ▼   Node
    ▼   Directory
          FileSystem
        File
      NodeList
      SimpleShell
  ▼   java.lang.Throwable
    ▼   java.lang.Exception
          Error
  ▼   java.io.Serializable
    ▼   java.lang.Throwable
      ▼   java.lang.Exception
            Error
  ▼   ShellInterface
        SimpleShell
```

Figure 4.8. Penumbra hierarchy view

Allowing students to experiment with the objects in the hierarchy view is key to improving their understanding of object-oriented programming. Eclipse provides the facilities to allow drag and drop features. I developed the PenumbraHierarchy-DragAndDropAdapter, which is the heart of this interactive feature. When a node of the hierarchy tree is selected, dragged, and dropped somewhere in the hierarchy view, the adapter will be invoked. The adapter will then determine the type of the source and the target. If a source class is dropped onto a class or an interface it will extend or implement that object. If the target is the empty space of the Penumbra hierarchy view, all extensions or implementations will be removed from the source class. Similarly, a source interface may extend a target interface. The underlying source files will be updated accordingly, all necessary import statements will be added, and the hierarchy view will be refreshed to display the change in the hierarchy. Furthermore, users can click on the objects in the hierarchy to open and edit the appropriate source files. This view is similar to the one provided in BlueJ, but it is more closely modeled after the design of Eclipse.

4.6 Help documentation

All help documentation in Penumbra is provided in HTML format. When the Eclipse help is selected, a web server is invoked that provides all the help contents. Furthermore, the default web browser will be started and directed to the web server to access the content. This solution is not the fastest, but it allows the addition of help content in HTML format, which provides for easy formatting and linking to other content.

To add help content to the Penumbra plug-in, all HTML pages have to be specified in a file called help.xml contained in the main directory of the plug-in. This is to integrate the files with all other Eclipse help content. Furthermore, I added a welcome page to the plug-in. This page will appear after Penumbra is being run for the first time. The page will point the student to the help pages for questions and also contains a link to the CS180 home page.

5 EVALUATION

Assessing the success of the Penumbra plug-in is difficult, as traditional, experimental methods cannot be applied. There are two main factors that prevent a completely objective assessment. Since we cannot collect measurements for Penumbra's success, we will have to rely on information obtained through experiences and surveys. Moreover, it is difficult to create a setting where we can expose one group of novice students to the default Eclipse, while we expose another group to Penumbra. We cannot do so in CS180 because all students have to be treated equally. This chapter will describe experiences collected through using Penumbra in CS180 and why I believe that the plug-in has been very effective in achieving its goals.

5.1 Evaluation by myself

The conclusions of this section are drawn from my teaching experience using standard Eclipse in the spring semester of 2003 and later using Eclipse/Penumbra in the fall semester of 2003. Furthermore, I have been supervising the use of Penumbra in Spring 2004. Throughout all of these semesters, I was the primary contact to be addressed with problems in the use of Eclipse and Penumbra.

In my experience, Penumbra has done a terrific job easing students into the use of Eclipse. Students had significantly less problems getting started with Eclipse and enjoying the benefits offered by Eclipse and Penumbra. When using the standard version of Eclipse in Spring 2003 a lot of time had to be spent on explaining how to set up and run projects. These problems have practically been eradicated with the help of the Penumbra project creation wizard and the check out functionality. Students can jump into their programming assignments with just a few mouse clicks. Moreover students enjoy the fact that they can easily install Eclipse and Penumbra at home, allowing them to work on their assignments from anywhere without much hassle. They do not need to be constantly connected to the Internet to be able to work on their projects. This is especially important, considering many students living off campus are using dial-up connections.

Furthermore, students are making use of the Penumbra hierarchy view, in order to better understand the layout of projects. Each semester we have two to three assignments that heavily rely on object-oriented concepts in which the view is helpful, such as inheritance and interfaces. In my opinion, the hierarchy view could be used even more by the teaching assistants in recitations in order to familiarize students with this concept.

It was noted to me by many students that the documentation provided with Penumbra was a tremendous help to get started with Eclipse. Students feel much more comfortable using the code assist features that are provided by Eclipse and highlighted by Penumbra.

However, there have also been some problems with the use of Eclipse and Penumbra. Our experience showed that Eclipse requires a fast computer in order to function efficiently. The designated lab had to be upgraded because the machines lacked sufficient amounts of main memory. However, this problem is not a flaw of Penumbra, but rather the Eclipse IDE. Also there seem to be rare occasions in which submission via CVS fails. This is unsatisfactory, and I am working to correct this problem. The submission via CVS relies on functionality provided by Eclipse, and so I can only make limited changes. It appears that the CVS metadata stored with the projects on the students' computers can become corrupted. Since we cannot monitor students' computers, it is difficult to track down the exact cause. Many fixes have been made to the CVS plug-in in the next version of Eclipse, and I hope they will help to completely resolve the issue.

A good indicator of Penumbra's success is the interest it has generated outside of Purdue. We received requests from colleges across the United States to use Penumbra. Parts of Penumbra have been integrated into a plug-in used in an introductory

Table 5.1
Survey Results – Fall 2003

Question	excellent	good	fair	poor	very poor
How would you rate the usability of Eclipse?	40 (27%)	81 (55%)	23 (16%)	3 (2%)	1(1%)
How would you rate the usability of Penumbra?	57 (39%)	68 (46%)	18 (12%)	2 (1%)	3(2%)
Question	**very easy**	**easy**	**fair**	**hard**	**very hard**
How difficult was it to set up Eclipse at home?	61 (43%)	47 (33%)	27 (19%)	8 (6%)	0 (0%)
How difficult was it to set up Penumbra at home?	57 (40%)	47 (33%)	29 (20%)	8 (6%)	2 (1%)

programming class at Stanford University. Also, the developers of GILD are interested to include some of Penumbra's features in their plug-in. Furthermore, the University of Toronto will be using Eclipse/Penumbra this summer. Besides these requests, Penumbra is downloaded several times a day from domains outside Purdue. We hope that making Penumbra openly available will spark even more interest.

5.2 Evaluation by students

As part of my efforts to improve Penumbra and evaluate the progress made thus far, I conducted surveys in CS180 in the Fall of 2003 and Spring of 2004. The response of the students toward the use of Eclipse and Penumbra was overwhelmingly positive. I asked the students to evaluate Eclipse and Penumbra on a scale from one to five, one being the best and five being the worst. The results for the two semesters can be seen in Tables 5.1 and 5.2.

The vast majority of the students thought that Eclipse and Penumbra together are a good development environment. Students thoroughly enjoy the code assist fea-

Table 5.2
Survey Results – Spring 2004

Question	excellent	good	fair	poor	very poor
How would you rate the usability of Eclipse/Penumbra?	32 (35%)	39 (42%)	18 (20%)	3 (3%)	0 (0%)
Question	very easy	easy	fair	hard	very hard
How difficult was it to set up Eclipse at home?	32 (35%)	38 (41%)	14 (16%)	6 (5%)	1 (1%)
How difficult was it to set up Penumbra at home?	32 (35%)	39 (42%)	14 (16%)	5 (5%)	1 (1%)

tures provided by Eclipse. Almost all students stated that they prefer Eclipse/Penumbra over any other environment they have used in their careers. Furthermore, they indicated that the installation of the environment was not a problem. The students wrote the documentation provided with the plug-in had helped them tremendously. I also asked the students to make suggestions for further improvements to Penumbra. A number of students would like to see check out and submit functionality added for labs. Overall, though the students did not have many ideas for innovations and seem to be satisfied with functionality offered by Penumbra.

6 FUTURE PLANS

6.1 Collaboration efforts

Our goal is to establish Penumbra as a preeminent environment for introductory programming. There will be other solutions available, but Penumbra should be able to evolve and attract a number of parties interested in using it. Consequently, we want to foster the use of Penumbra at other colleges and universities. We are not only interested in the use of the plug-in by other institutions, but also in the feedback we would get, in order to help us further improve Penumbra. In order to facilitate this aim, we have already made Penumbra accessible on the Eclipse Community Education Project (ECESIS) web site [16].

Furthermore, we want to encourage other groups to actively contribute to the Penumbra project. Currently, all development is performed on a CVS server at Purdue. We already allow for anonymous read access to the repository, so that other groups can download the source code and adapt it to their needs. The ultimate goal though, is to create a publicly available CVS repository that allows for development by more than one person.

6.2 Adaptation

Currently, a major release of Eclipse is under development. Version 3.0 is still in the beta phase, but a stable version should be available by the end of June. Drastic changes have been made to the Eclipse interface, as well as the underlying engine [17]. The new version of Eclipse also features technology that will make it easier to precisely craft perspectives, which includes more sophisticated functionality to hide and disable existing Eclipse features. Due to the improved perspective creation process, as well as the fact that Penumbra makes heavy use of Eclipse technology, it will be necessary to adapt our plug-in to the new version of Eclipse. During this summer, I will develop a version of Penumbra that can be used with Eclipse version 3.0 for the Fall semester of 2004.

6.3 Feature ideas

Through the use of Eclipse and Penumbra in CS180, as well as my exchanges with other groups and educators, I have collected a set of ideas that are worth being added to Penumbra. However, integrating new features in our plug-in will again add to the complexity of the perspective we want to present to the students. We will need to carefully determine how we can add these features while keeping the complexity at a suitable level.

6.3.1 Lab submission

Check out and submission of projects have proven to be very effective because students do not need to switch to the command line or use a web browser to get started. In addition, the process works fast and seamless. In CS180 at Purdue University, we make a distinction between projects and laboratories. There are eight projects that students have to work on throughout the course of the semester. The students are given one to two weeks to complete the projects, based on their difficulty. Laboratories on the other side are given weekly, and are to be completed within a two hour time frame. Since the quantity of students prevents us from having all students take the laboratory assignments at the same time on the same day, students that complete the assignment at a latter time, might gain an advantage over their peers. Hence, we make the assignments only available for each student respectively, while he or she is in a laboratory session. We do this by having the students download the assignments from a website that has restricted access based on the students login. After the students download the skeleton files for the laboratory assignment, they have to import them using the default Eclipse import features. After the assignment is completed, students have to submit their solution using the turnin command.

This is a non-optimal solution for several reasons. Students have to switch to the command line and use tools they are not familiar with. Moreover, I experienced that the students are confused as to why they have to apply two different means of check out and submission for projects and laboratories. From a user's perspective, there is no understandable reason why we cannot develop a unified solution to the problem. In addition, relying on a turnin command makes the solution dependant on a certain system architecture. An ideal solution to the problem should work on any platform supported by Eclipse, without the user noticing.

However, such a solution is not easily implemented with CVS. We need to find a solution that is completely reliable and very easy to manage. At this point, using a CVS repository would require us to manually grant and remove the students' permissions to access certain modules. Our solution will have to be more user-friendly, such that it can be managed with minimal effort.

6.3.2 Simplified debugging perspective

An important part of being a good programmer is to be able to find one's bugs in one's programs. In order to facilitate this process, there are many debugging tools available, and Eclipse also contains a sophisticated debugging perspective. Unfortunately, debugging is often overlooked in course syllabi, and students feel discouraged to use these tools because their interfaces seem unintuitive or complicated. Consequently, a substantial number of students will make it to higher level computer science classes without being familiar with the use of debuggers.

This shortcoming has been addressed by the developers of GILD, DrJava, and BlueJ. All these environments contain simplified debugging features in order to encourage students to become more familiar with the process. DrJava and BlueJ include separate debugging perspectives, while the debugging features are integrated into the default GILD perspective. We are hesitant to include debugging features in the default perspective because students are unlikely to use them at the beginning. At this point, it seems more beneficial to create a separate debugging perspective. First, we need to evaluate the progress made in the next version of Eclipse to see what improvements are still necessary.

6.3.3 JUnit

Software testing is an integral part of professional software development, yet relatively small amounts of time are spend on this matter in colleges or universities. Students often submit projects for grading that are tested insufficiently, just to find that they receive unsatisfactory grades. Part of this problem is that students do not spend enough time on testing. Another part is that students often do not know how to test software properly. A popular tool for testing is JUnit, "a simple framework to write repeatable tests" [18].

JUnit allows the developer to quickly design test cases that can be used easily to efficiently test software. In fact, we have used JUnit in CS180 to test and grade students' projects. Eclipse comes with a JUnit plug-in that allows the developers to test their software. It should be our goal to integrate functionality into Penumbra that will allow students to test their code using JUnit. We could provide students with test cases that we can package with the skeleton projects. Students need to be able to run these at a click of a few mouse buttons. Moreover, we should encourage students to write their own tests. Consequently, it might be necessary to simplify the existing JUnit plug-in interface, as to allow the students to see the benefits of this process.

6.3.4 Penumbra perspective for teaching assistants

So far our focus has been entirely on providing the novice student with an easy-to-use development environment. However, this does not have to be the only perspective provided by Penumbra. The developers of GILD have already gone so far as to provide the instructors and teaching assistants with their own specialized perspective. There are in fact several reasons that favor the development of such a perspective. Often times instructors are hesitant to use Eclipse because they are more familiar with other tools. However, it would be beneficial for the instructors to use Eclipse because of the wealth of features it can provide. The instructor's perspective included in GILD is primarily used to allow for easy management of course materials. The instructor can write and post all course content from within Eclipse. This is valuable because the instructors can create links between their explanations and problems in programming assignments.

Furthermore, our perspective could be used to allow the instructor to develop project assignments, including documentation, skeleton files, and test cases, entirely within Eclipse. The instructors should then be able to quickly post the assignments directly to the CVS repository and web server. The perspective could also facilitate the development of JUnit test cases that could then be used to grade the students projects.

7 CONCLUSION

My work shows that Eclipse can be modified to be an excellent development environment for introductory programming courses. The simplified Penumbra perspective, as well as the added features such as check out and submission, have increased the acceptance by the students. Furthermore, the development of features such as the Penumbra hierarchy view increases the pedagogical support by Eclipse.

In my opinion, Eclipse/Penumbra can be made an even more suitable environment through the implementation of the ideas presented in the previous chapter, although we need to be careful to retain the simplicity of Penumbra while adding new features.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Sherry Shavor, Jim D'Anjou, Scorr Fairbrother, Dan Kehn, John Kellerman, and Pat McCarthy. *The Java Developer's Guide to Eclipse*. Addison-Wesley, 2003.

[2] Michael Kölling. Teaching object orientation with the Blue environment. *Journal of Object-Oriented Programming*, 12(2):14–23, 1999.

[3] Eric Allen, Robert Cartwright, and Brian Stoler. DrJava: A lightweight pedagogic environment for Java. In *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*, pages 137–141. ACM Press, 2002.

[4] Andrew Patterson, Michael Kölling, and John Rosenberg. Introducing unit testing with BlueJ. In *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education*, pages 11–15. ACM Press, 2003.

[5] Michael Kölling, Bruce Quig, Andrew Patterson, and John Rosenberg. The BlueJ system and its pedagogy. *Journal of Computer Science Education*, 13(4), 2003. Special issue on Learning and Teaching Object Technology.

[6] Charles Reis. A pedagogic programming environment for Java that scales to production programming. Master's thesis, Rice University, 2003.

[7] Erik Sandewall. Programming in an interactive environment: The "Lisp" experience. *Computing Surveys*, 10(1):35–71, 1978.

[8] S. Hillion. DynamicJava. http://koala.ilog.fr/djava.

[9] Margaret-Anne Storey, Mary Sanseverino, Daniel German, Adrian Damian, Jeff Michaud, Adam Murray, Rob Lintern, James Chisan, Marin Litoiu, and Derek Rayside. Adopting GILD: An integrated learning and development environment for programming. Available at http..., 2003.

[10] Margaret-Anne Storey, Daniela Damian, Jeff Michaud, Del Myers, Marcellus Mindel, Daniel German, Mary Sanseverino, and Elizabeth Hargreaves. Improving the usability of Eclipse for novice programmers. In *Proceedings of the 2003 OOPSLA workshop on Eclipse Technology eXchange*, pages 35–39. ACM Press, 2003.

[11] Charles Reis and Robert Cartwright. Taming a professional IDE for the classroom. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*. ACM Press, 2004.

[12] Frank Mueller and Antony L. Hosking. Penumbra: An Eclipse plugin for introductory programming. In *Proceedings of the 2003 OOPSLA Workshop on Eclipse Technology eXchange*, pages 65–68. ACM Press, 2003.

[13] Walter Savitch. *Java: An Introduction to Computer Science and Programming, Third Edition.* Prentice-Hall, 2004.

[14] Concurrent Versions System. http://www.cvshome.org.

[15] Erich Gamma and Kent Beck. *Contributing to Eclipse.* Addison-Wesley, 2003.

[16] Eclipse community education project: An Eclipse technology research subproject. http://www.eclipse.org/ecesis/.

[17] Eclipse project draft 3.0 plan. http://www.eclipse.org/eclipse/development/eclipse_project_plan_3_0.html.

[18] JUnit. http://junit.sourceforge.net/.