Improving the Usability of Eclipse for Novice Programmers

Margaret-Anne Storey[1] Jeff Michaud[1] Marcellus Mindel[2] Mary Sanseverino[1]
Daniela Damian[1] Del Myers[1] Daniel German[1] Elizabeth Hargreaves[1]

University of Victoria, BC, Canada[1]
and IBM, Canada[2]

Abstract

This paper describes how we are designing a set of plug-ins to improve teaching and learning of Java programming. Based on requirements gathered from key project stakeholders, the plug-ins include perspectives for both students and instructors. These plug-ins are being developed as part of the Gild project. The paper summarizes our design process from requirements gathering through to initial tool evaluation. In particular, it details the student perspective and describes how it addresses many pedagogical issues facing computer science educators today. Gild is currently deployed and in use at the University of Victoria.

1 Introduction

The Eclipse Java Development tool suite is rapidly becoming the integrated development environment (IDE) of choice for many programmers [1]. Many instructors are also starting to use Eclipse as a tool to help with teaching in the classroom and are recommending Eclipse to their students. Eclipse's sophisticated features (such as the scrapbook and code assist features) support key "teachable moments" and allow instructors and students to more easily demonstrate (explore) the Java language and object-oriented concepts.

Admittedly, Eclipse has many powerful features that can assist both expert and novice programmers and Java teachers. However, we believe that Eclipse can be improved to offer enhanced support to novice programmers as well as instructors. Firstly, the Eclipse user interface may be overly complex and that this complexity could pose a barrier to learning. Secondly, improving Java instruction through Eclipse requires offering support for features that are commonly available in web-based learning tools.

Currently, the teaching practice in our community is to use a separate web-based learning environment (or other tools for managing course content). This removes the students and the instructors from the programming environment that is the cornerstone of their learning. Although this may be a familiar issue in other disciplines, we and other researchers in Computer Science Education believe it is a particular problem for programming courses [4]. The artefacts that are studied in programming are abstract and dynamic and hence are difficult to perceive without tool support. For these reasons, the need to closely integrate course material describing theoretical concepts with practical examples is particularly important for programming courses.

Due to Eclipse's highly flexible and customizable environment, some may suggest that it is simply a matter of customizing a perspective so that it provides a simplified view for novice programmers or instructors. Unfortunately, the difficulty is determining what such a simplified view should look like and if there are new features that could provide more learning support.

In our project, called Gild [6], we are carefully considering how to create a simple and yet powerful development environment to help students learn more effectively and instructors teach Java more effectively. Our goal is to leverage the existing sophisticated features offered by Eclipse and also consider how features offered by other Eclipse plug-ins and pedagogical tools for teaching programming concepts (such as DrJava [2] and BlueJ [3]) can be combined to improve teaching and learning.

During the initial phases of the Gild project, we have focused on gathering requirements from students, teaching assistants and instructors for an integrated learning and development environment for Java. Such an environment should address the needs for better teaching and learning Java as outlined above. Using this set of requirements, we have designed a set of plug-ins (collectively called Gild) that provides two perspectives – one for instructors and one for students. The instructor perspective provides support for managing a programming course, and creating course content that is closely integrated with code examples. The student perspective presents a simplified development environment that has additional features to help them easily navigate between course material

and program examples. In this paper, we focus on the student perspective.

In Section 2, we summarize the requirements gathering approach we have been following. Although we cannot describe our requirement gathering activities in detail, we provide a summary to give the reader some intuition about the validity of the requirements gathered. In Section 3, we present the initial prototype and summarize how it differs from the default Eclipse view. In Section 4, we present some preliminary results from a small user study we conducted to evaluate the usability of the Gild student perspective and discuss our ongoing implementation and evaluation efforts.

2 Eliciting Requirements

Through a variety of research techniques, we have been eliciting requirements for the Gild plug-ins by collecting data from four groups of stakeholders: students, teaching assistants, course instructors and lab administrators. We have deployed questionnaires to student programmers, conducted focus groups with teaching assistants and instructors both at the University of Victoria and Dalhousie University, and finally conducted interviews with lab administrators at the University of Victoria and at the University of British Columbia. Requirements gathering is ongoing as we conduct user experiments and deploy preliminary versions of Gild in classroom settings. The Gild website [6] lists the questionnaires and focus group questions we have used.

Throughout all of the requirements gathering efforts, one of our main objectives has been to understand how an integrated learning and development environment could improve how students learn how to program in Java. As a first phase, we determined the essential features that should be in a Java editor to support programming, and secondly, we determined requirements for additional features which would provide cognitive support for specific learning activities and needs.

The focus groups and questionnaires have been very successful in helping us identify the desired features for an integrated learning and development environment. However, these techniques also revealed that the ideal characteristics of a programming environment for novice students are a matter of philosophical debate which is highly dependent on course objectives, instructor preference, and student ability. For example, there are certain mechanisms in an IDE that some instructors feel should be exposed to a learning student, while others feel the same mechanisms should be hidden. This leads us to the requirement to have some features be configurable to suit different teaching and learning styles.

In the next section we describe how the student perspective is being designed to realize the requirements we have so far identified.

3 Gild Student Perspective

In order to create an integrated learning and development environment for Java, we are leveraging Eclipse as the underlying technology.

The requirements we have collected thus far have helped us distinguish which features in Eclipse are essential for student programmers, and which features are unnecessary and should be excluded as they may introduce additional cognitive overhead. Furthermore, we have identified which features need to be simplified for Gild and we have considered new and enhanced features that could provide additional cognitive support for learning activities. Finally we deliberately incorporated sufficient flexibility to accommodate the broad range of pedagogical approaches that were evident from the gathered requirements.

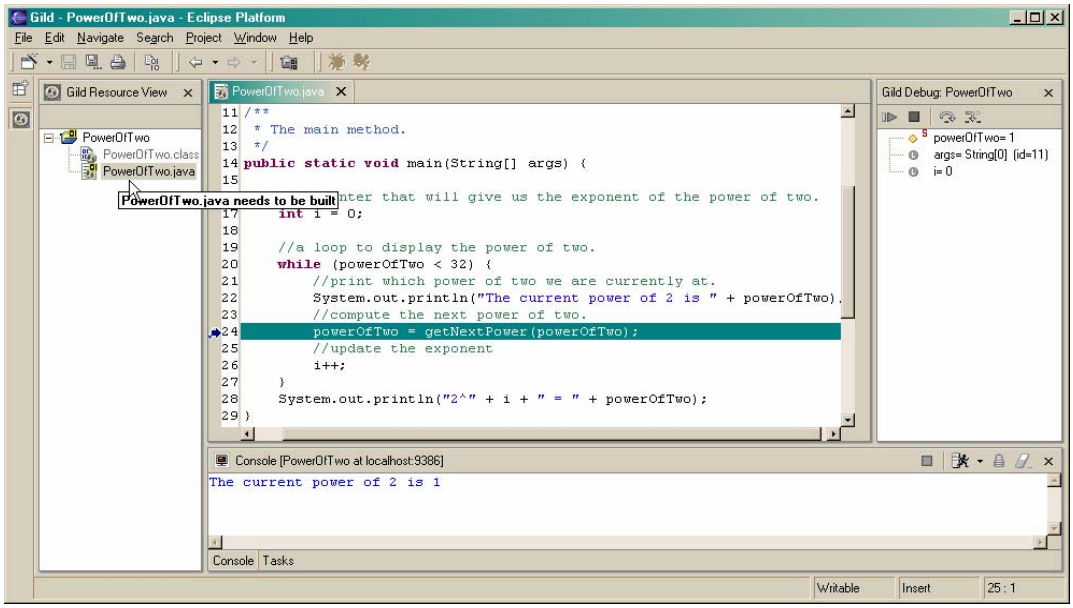
3.1 Core Features to be *included*

From the focus groups with instructors and teaching assistants, we determined that Gild should include a basic file editor with syntax highlighting and line numbers. The console, task and resource views were recognized to be needed for managing resources, tasks and output. The debugger was also seen as an essential feature by many of the instructors and teaching assistants – but not all agreed with the use of a debugger. The project import and export features were also requested. Fig. 1 shows the student Gild perspective.

3.2 Eclipse features to be *simplified*

Some features are easy to simplify and involve varying the default settings for the Gild plugin from those in Eclipse. Other changes involve more effort as we have to programmatically remove features and configure views.

By default, the Eclipse Java Editor does not show line numbers but line numbers were deemed to be an important aid for learning and hence are present in the Gild Java Editor by default. The task view was also seen as important



but we changed the default filter to only display errors relating to the current project (as novice programmers tend to consider only one project at a time).

In our focus group discussions, there was very strong support to separate the save, build and launch actions. The main concern is that students must learn that "files need to be compiled" and that the IDE should not do it "behind the scenes". Consequently we configured Gild so that these operations are distinct operations. However, not every instructor agreed with this, and some felt that the convenience of the automatic build offered by Eclipse was more important. We address this philosophical difference by having specific Gild preferences (see Section 3.6).

The instructors raised concerns that novice programmers often have difficulties figuring out how to compile and run programs in an unfamiliar IDE. Thus we strove to achieve simpler launching actions in Gild. Currently Eclipse's launch buttons provide options for multiple JRE's, source look-up paths, command line parameters, etc. Novice programmers may get confused by these options, so the Eclipse launch commands have been replaced with "one click" Gild run and debug commands. Default configurations are automatically set and a dialogue box is presented if there are multiple main methods or applet classes in the selected project to launch from.

Some instructors and universities require that first year students use a debugger extensively and hence this became a key requirement. Unfortunately, the debugger in Eclipse is quite complex. Considering the needs of novice programmers, we provided the following features in the Gild debugger: 1) it is integrated into the Gild perspective as a set of views, so that the user does not have to switch context; 2) it only runs one thread, hiding the concept of threads from novices; 3) it displays the current state of the stack frame (as a variable view); 4) it restricts the debugging actions to resume, terminate, step over, and step into; 5) it only suspends execution within code that the user has placed in the workspace. Otherwise, the debugger acts similarly to the Eclipse debugger.

The requirements also revealed that students face difficulties managing their files in their development environment. Consequently, this led to the development of a simplified import and export feature for projects and files. We predict that these tools will make it easier for students to transport code to and from home, make backups, submit their assignments, as well as interact with sample code provided by instructors.

3.3 Eclipse features to be *excluded*

We further simplified the interface by removing features deemed as non essential from the menus and toolbars. For example, we removed wizard buttons for creating classes and interfaces. These features are a convenience but do not necessarily help new programmers learn how to program as they may offer too much "scaffolding". Students

have to know how to write code on an exam with no compiler available. We also decided to remove refactoring features as they are unlikely to be useful to or comprehensible by novice programmers.

3.4 Eclipse features to be *enhanced*

To improve the cognitive support and usability offered by Eclipse to students, some enhancements were required. For instance, the teaching assistants indicated that students often get confused about the state of their files. We looked to well-accepted user interface guidelines to guide us on how we could improve the user interface. Don Norman lists four key user interface principles of feedback, visibility, mappings and affordance [7].

Using Norman's principles as counsel, decorators and tooltips were added to the file icons in the resource view to make the file status visible and to provide feedback on necessary actions (see top left of Fig. 1). Such cues provide cognitive support for the student programmer and prompt them to take the required actions before trying to launch a program. These cues also help to reinforce the 3-step process of save-build-launch. It was also observed that students get confused about the difference between `.java` and `.class` files. To address this issue, we gray-out `.class` files in the resource view. If the student does try to edit a `.class` file, text is displayed to remind the student of the difference between class and java files, while providing a summary of the selected class.

3.5 Features to provide Cognitive Support for Teaching and Learning

So far we have focused our discussion on how we are creating a simplified and more usable development environment for novice programmers. In addition to these changes, we also need to incorporate features from web-based learning tools. So far in our project, we have added the ability to more easily navigate between course notes and the code examples within Eclipse.

In Gild, when the student imports a project, it should include notes written in HTML. We integrated a simple HTML browser as a view within Eclipse. The browser has some custom features to support navigation between course notes and program code. First, it supports links that will direct Eclipse to open up specific files in the editor and scroll to the appropriate code. Secondly, segments of code from project files (such as Java files) can be dynamically included into the HTML document, making it possible to keep notes synchronized with example code. [1]

The instructors and teaching assistants stress that it is important to have a tutorial so that students can learn how to use Gild easily. We have written a user document to help students learn the environment (see our website [6]). These files are packaged in the release and can be browsed within the HTML browser we have added. Links to the source code and embedded code snippets are included in the tutorial using the HTML feature we just described.

3.6 Support for different Teaching Philosophies and Learning Needs

In order to support different teaching philosophies, we added a separate Gild preference dialog to indicate which features are conceptually important to the Gild perspective. These preferences may be explicitly changed by instructors (before students download Gild for a particular course) or students may themselves change these features to suit their expertise and learning style.

The set of features that we felt were most essential to Gild included the ability to do automatic builds; the filters used to direct Debug stepping; and the use of CVS version control. Version control is favoured by some instructors for first year, but not by others. Since most did not use it, we turned this feature off by default.

Students and instructors are of course still at liberty to include or exclude other features from Eclipse as they deem necessary. This approach in our design means that Gild is more like a set of training wheels for a bicycle then a tricycle, and that over time students can transition to the full feature set available in the Eclipse JDT as their expertise and comfort level increases.

3.7 Summary

Table 1 provides a summary of the changes we made to Eclipse to achieve the Gild plugin.

4 Evaluating Gild

In early August of 2003, we conducted an initial user study of the Gild Student Perspective with three student participants. One participant was a high school student and new to Java, the other two

[1] The instructor has to explicitly add tags to the HTML course notes to enable these new features.

Category	Feature
Core Features to *include* from Eclipse	Syntax highlighting Line numbers Compile and launch operations Debugger Task view Resource view Project import/export
Features to *exclude*	Wizards Refactoring
Features that are *simplified*	Launch: one-click Debugger: single thread, step over Project import/export simplified Task view: changed default behaviour to show only errors for the current project Line numbers: on by default
Features that are *enhanced*	Resource View: decorators on resources to indicate file state and distinction between .class and .java files
New features to support *learning* needs	HTML browser with new features to link to and dynamically include source code
Support *different teaching and learning styles*	Configurable settings for Gild: Use of debugger, debugging filters, CVS, automatic build.

Table 1: Summary of features in Gild

were just finishing a second university Java programming course. All of the participants were somewhat familiar with IDEs (Net Beans or Visual Studio). In this initial user study we concentrated on ease of use and user interface issues. We did not test every feature of the Student Perspective.

Testing did yield some useful results. It identified interface issues (e.g. button placement, API linking). It also identified user confusion between the edit and debug modes. This may be due to the fact that the debug view is no longer in a separate perspective. Future releases will address this problem though the use of an appropriate interface cue. The students were impressed by the debugger but the more experienced students did not use it for the simple tasks.

Overall, the participants indicated that the Student Perspective was easy to use. After testing, one participant said "I am going to go home and download Eclipse".

The testing protocol used in this initial study is available at our website [6]. Currently we are recruiting participants with different levels of programming experience to fully test all features in Gild. We are also recruiting students who have no exposure to IDEs.

This preliminary version of Gild has been deployed in the computer labs at the University of Victoria. It will be used during two courses by students and instructors in the Fall 2003 term. We will collect feedback and improve it accordingly during this term. At the same time we will be evaluating the instructor perspective.

In the meantime, we are exploring the pedagogical benefits of other features in Eclipse such as code assist and local history. We are also prototyping other features that were recommended as a result of our requirements gathering efforts. Collaborative features common in web-based learning are also being considered. For instructors, we are exploring the incorporation of visualizations and animations for teaching OO concepts. Finally our other work involves creating *workbooks* for the students where we will leverage the Help system in Eclipse.

The latest Gild plug-ins can be downloaded from [6]. We welcome any feedback as we continue to test and develop these plug-ins.

Acknowledgements

We wish to acknowledge participation from other group members working on Gild who have made this work possible: Marin Litoiu from IBM, Karen Parker from Dalhousie University, Derek Rayside from MIT and Adrian Damian from UVic.

About the Authors

Margaret-Anne Storey, Daniela Damian, Mary Sanseverino and Daniel German are faculty members and Jeff Michaud, Elizabeth Hargreaves and Del Myers are research assistants at the University of Victoria. Margaret-Anne Storey is a winner of an Eclipse Innovation Award from Fall 2002. Marcellus Mindel is a manager at IBM Canada.

References

[1] Object Technology International Inc, *Eclipse Overview*, 2003 http://www.eclipse.org/ white-papers/eclipse- overview.pdf

[2] E. Allen, R. Cartwright, and B. Stoler, *DrJava: A lightweight pedagogic environment for Java*, September 7, 2001, Presented at SIGCSE 2002.

[3] D. Barnes and M. Kölling, *Objects First with Java A Practical Introduction using BlueJ* Prentice Hall / Pearson Education, 2003,

[4] M. Daniels, M. Petre, and A. Berglund. *Building a rigorous research agenda into changes to teaching*, Proc. of the 3rd Australian conference on Computer science education. p. 203-209, 1998.

[5] D. A. Norman, *The design of everyday things*, New York: Double Day, 1990.

[6] Gild website: http://gild.cs.uvic.ca.