GILD: What we have learned

Introduction

Novice programmers are often given the challenging tasks of not only learning programming concepts, but also learning a programming language and a development environment at the same time. While Integrated Development Environments (IDEs) are powerful tools for software developers, they are designed with professional programmers in mind. For a novice programmer, the wide range of options and features available can be overwhelming. However, simplifying the IDE is not enough. There is the opportunity to leverage the compiler and the other aspects of an IDE to provide pedagogical support towards learning the fundamentals of programming. In 2003, the Gild project was envisioned to provide the cognitive support of an IDE, but with a simplified feature set.

Gild has gone through a number of iterations. Over the past three years, Gild has been used by over 400 students at the University of Victoria, as well as by students in the United States and Germany. Each time Gild has been used, we have engaged in extensive evaluation, including surveys, focus groups, interviews, ethnographic studies and user studies. This document aims to summarize some of the insights and lessons we have learned along the way.

While our experiences are drawn primarily from our studies of the Gild IDE, many of our findings point to changes that affect users of Eclipse. We have followed users from their first experiences with an IDE through to their first usage of more advanced features available in an IDE. The following lists some of our key observations. Further observations and details are available on the Gild Website (http://gild.cs.uvic.ca).

Installation can be difficult

The installation process is the first interaction a user has with the software tool. It will therefore form a user's first impression of the tool and inform his or her overall sense of it. In initial survey responses, students were split over the ease of the Gild installation process, which really involved three different installations: a Java SDK, the Eclipse IDE, and finally the Gild plug-in. For students who had difficulties, some responded that the order and number of installation steps were confusing, as many did not understand the dependencies between these steps. Others had computer systems that could not meet the memory requirements of Eclipse.

Student confusion resulted primarily from having to download files from multiple websites. As a result, it was recommended that students be provided with a single file to download. This file would include all the necessary components for the installation. An installer was also recommended to help guide students through the installation process. It was hoped that this would also ensure that students were using the same or similar versions of Eclipse, Gild, and Java – which is useful in a teaching environment.

Before the Spring semester of 2005, a single Gild installation package was created for Windows using the Nullsoft Scriptable Install System. This package included the Java SDK, Eclipse, and Gild. After the deployment of this downloadable, the number of e-mails sent to Gild Help asking

for support regarding installation difficulties were reduced, and student opinion of the ease of install improved significantly (see Fig. 1).

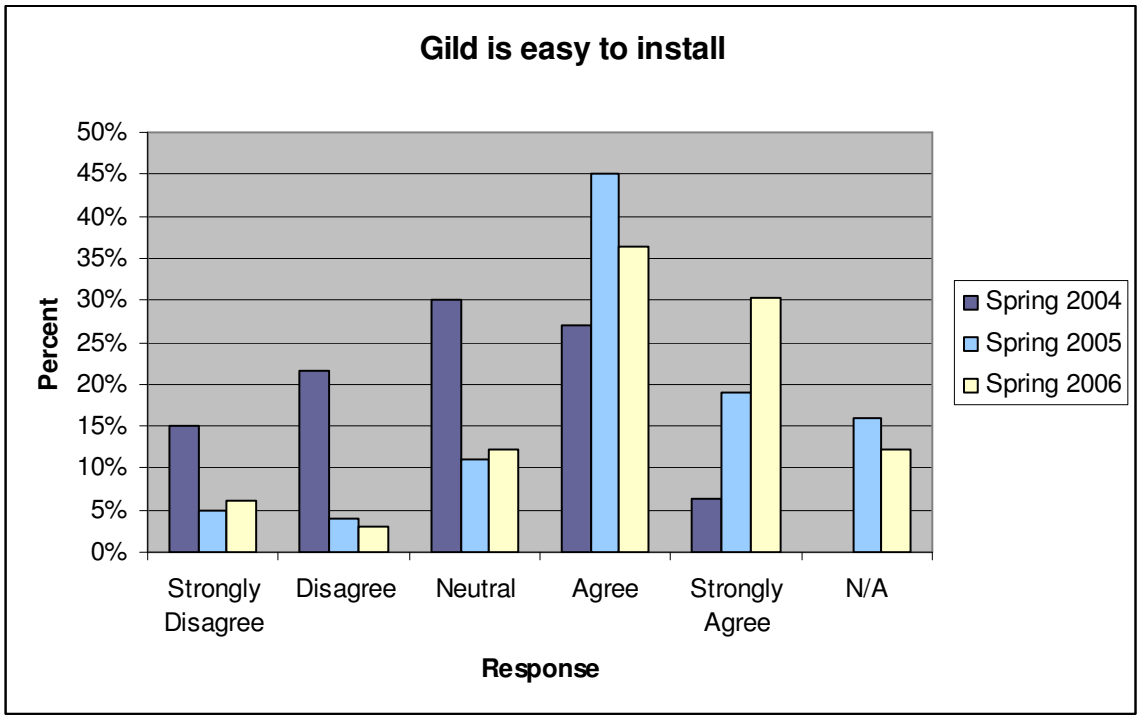

Figure 1 –Comparison of ease of install before and after a single installation download was created for Gild users for use in Spring 2005.

File creation is not ideal

"There are a few things that are not quite as simple. Example, creation of a file was not totally intuitive. It is strange that you have to create a project first." - CSC 115 student, February, 2004

The project/file creation process was the most disliked feature in Gild, according to students in the Spring 2004 survey. Although there is a simple file creation wizard in Gild in which students may simply enter a file name and associate it with a folder/project, students complained that a project must be created first. Many students who use Gild had previously used a simpler program called Textpad in which there is no concept of projects. So the idea of a project was not intuitive to them.

Early on in the Gild project, we attempted to solve the confusion by adding the concept of a "default" project in which all files that were created or opened without being associated with a project in the workspace would be saved. We originally interpreted the confusion as being due to the fact that students were unable to open files that existed outside of the workspace. So, when Eclipse added this functionality as a standard feature, we removed the "default" project. However, some confusion still existed because students could not create files within Eclipse without first having a project to create them in. So, the concept of a "default" project may still be needed in order to facilitate the adoption of the project paradigm for resource management.

Import and export requires simplification

"Worse that [sic] JDK/SDK due to that accursed Projects system. I keep files plenty well organized on my hard drive: I don't want to reorganize tham [sic] to use with Gild, especially when gild makes it confusing which version of a file is stored where." – CSC 115 student, February, 2004

"Simplified export/import/save; perhaps put all the export options on a single window; save project auto exports to designated directory." – CSC 115 student, April, 2005

A student's mental model of files and the filesystem is often based on their previous experience working with editors, such as Textpad, Word or Excel. With an editor, students are working directly on the file they open. Eclipse uses a different model, taking over the management of the filesystem by using a workspace. When students begin working with Eclipse/Gild, they are presented with the unfamiliar option to "import" a file. Student's mental models often associate this option with the "open" option. This leads to confusion as they incorrectly believe that any changes made to their imported files will be reflected in their original files. Letting Eclipse manage the workspace is not the problem, but students could benefit from receiving more knowledge of the location of their workspace and the files contained within.

Much of the confusion about importing and exporting projects and files may be due to the fact that in our situation at the University of Victoria, we had no option but to force the students to use it as their primary method of storing their projects. At the university, students work at workstations in various computer labs and are given access to a limited amount of personal hard-disk space on a networked drive on a server. All data on the local workstation is cleared during nightly jobs. Confusion may have been avoided if it were possible for the students' workspaces to be saved on this network drive instead of on the local workstation. However, this was not an option for two reasons: 1) even short blips in the connectivity of the network would cause Eclipse to lose its reference to the workspace, causing data loss and; 2) the size of a workspace would often exceed the students' personal data limit because of the fact that plugin settings are saved with the workspace. If these problems were addressed, much of the confusion could have been avoided.

The process to import a project is cumbersome and it is too easy to become confused, especially for a novice. Students at the University of Victoria are often given an archived project to import into their workspace. Using Eclipse, students often get confused between importing an archived file and an existing project. Partly this is caused by how files are exported. Students can export either projects or files. The result is an archived file. If students, for example, choose to import an archived file that contains a project into an existing project, they will end up with a nested project. This creates package naming problems, which take a long time for students to correct. In cases such as this, an auto-correct feature for package naming would be beneficial. With respect to this situation, it is unfortunate that Gild removes the auto-correct feature.[1]

[1] In designing Gild, decisions were made to remove certain features of Eclipse to simplify the user experience. As the focus of Gild is to help novice programmers *learn* how to program and use an IDE, having automated features, such as auto-compilation, were thought to be counterproductive, as the problems could be solved without any understanding. For instance, auto-compilation is initially turned off to emphasize that code must first be

The simplified debugger is useful

"The debugger has been very useful. It is probably the single most important feature in Gild" – CSC 115 student, February, 2004

The simplified debugger is one of the most popular features in Gild. The Eclipse debugger has a handful of views that are organized as a perspective. Usually, when the programmers want to use the debugger they switch into this other perspective. With Gild, we wanted to avoid switching away from the Gild perspective, because doing so could disorient new programmers[2]. So, in Gild, a single view was designed which simply displayed the current stack frame (as in the Eclipse Variables View), and allowed the user to step through source code. The Eclipse Breakpoints View was also included with the Gild perspective. These two views are not visible at all times, but only when debugging occurs.

The Gild debugger is single-threaded to reduce complexity. As novice programmers normal work only on single-threaded programs, the multi-threaded Eclipse debug view is unnecessarily complex. The stepping features of the Gild debug view were also designed to focus only on the variables and objects that are relevant to the files being written by the programmer. Most novices are trying to debug specific problems with their own code. Hiding irrelevant source such as that for the Java class libraries helped them to accomplish this task by allowing them to focus only on their code.

Another suggestion that we have received (from instructors and teaching assistants (TAs)) is that it would be nice to have visualizations of the objects that are created, and if users could see the changes in objects over the execution of the program. This would be like a recording of the program execution. The visualization could show the relationship between objects and object instances.

"The debug feature is slightly difficult. I never know exactly what's happening." – CSC 115 student, February, 2004

Although the debugger in Gild is simplified, students still need some guidance. How to use a debugger is not necessarily taught in a course, and students need to be aware of what types of errors the debugger is useful for helping to solve (not compiler errors!) Once students understand that a debugger is for stepping through code as it is executing, most find it very useful. The debugger in Gild automatically sets a breakpoint on the first line of execution. This automatic breakpoint is useful for students.

Some students have reported using either Textpad or Eclipse, but returning to Gild when they encounter a problem that requires a debugger.

---

compiled/built before it can be run. However, typically instructors tell their students to change this setting very quickly.

[2] We observed students having problems switching between the Java perspective and Debug perspective during a user study involving Eclipse and Gild. For more information, please see the "Study of Novice Programmers using Eclipse and Gild".

Error notification, extra help and navigation is useful

"better error messages... error on string tokenizer :/#%@^ expected, what the hell does that mean, honestly" – CSC 115 student, February, 2004

The error notification and navigation features in Gild, which are inherited from Eclipse, are another popular feature. Students have commented on the usefulness of the red squiggly underlining of an error's location in the code, and also the linking from the console to the error.

However, even with this form of error notification, students struggle to interpret the cryptic error messages they are presented with. Novice programmers need to learn how to approach these errors. To aid with this, Gild added extra error support for common errors[3]. The extra error help feature in Gild was created to provide students with more information about the possible problems, and some suggested solutions. The original compiler message is still displayed as usual, so novices can make the mental link between the error and how they solved it.

Less is better (Keep it simple)

"The Gild interface is far simpler to use than the Eclipse interface. I feel like the simplified right click menus help me with finding the specific function I want to use faster, instead of swimming through menu after menu when using Eclipse." – CSC 115 student, February, 2004

Unnecessary menu options or windows provide clutter and confusion for students. Eclipse has many advanced features for expert programmers; novice users do not need to be intimidated by a seemingly over-complex tool. The students at the University of Victoria are often transitioning to Gild after using the Textpad program for programming assignments. Textpad provides little more than syntax highlighting and short cuts to compile and run programs through the DOS prompt.

Integration provides structure for instructors

Instructors who have used Gild in the classroom at the University of Victoria have found Gild to be useful for helping to organize their lecture material and examples. Instructors liked that they could create a Gild project (zip file) that contains PowerPoint slides, html files, and java files, and make these projects available for students to import into Gild or to open using an unzip utility. This not only makes it easy for posting material, but it also helps instructors organize materials for future terms. Well organized projects also help students keep their java files associated with the programming concepts that are being taught. Additionally, this provides an easy way for students to get their feet wet, practicing with the code.

Many instructors like to run code examples while teaching. Unfortunately, the standard font size used in Gild for the code editor is too small to be nicely displayed on a projector for the entire class to see. Teachers would like to have a "presentation mode" so that the text and console are visible to students in a classroom setting (using the projector).

[3] For more information, please see "An Exploratory Study of Novice Programming Experiences and Errors".

Better visibility of features is needed

Where do new users begin? The Gild tutorial is located in the Help menu, which few students appear to access. Through observation we have found that students do not spend time exploring the IDE after installing it. Students who are learning how to use an IDE are also learning a programming language (and associated concepts) at the same time. Students want a tool that will support their learning and will not require much time to learn how to use the tool. It is important that the tool uses language that the students are familiar with, and that new concepts are explained clearly.

Some students who used Gild were not aware of all of the features available. The debugger and extra error help features were both designed to assist students, and are not hidden away in the menus. The Gild tutorial, which can be found by looking at the Help section in the Help menu, may be difficult for students to find.

The environment should evolve with the learner

While many users are new to programming and IDEs, some people bring with them previous experience and are able and ready to use more advanced features.

Gild is intended to act as a stepping stone to help students become familiar with the Eclipse IDE. The developers of Gild tried to determine the best set of features to fulfill this goal. However, students have different backgrounds and skill sets. As such, some students may need more support, while others are able and ready to use more advanced features. Ideally, the environment should evolve with the needs of the user.

An example of this involves the auto-compilation feature in Eclipse. In Gild, this feature was disabled, forcing students to build their code before running it. Forcing this process helped to teach the concept of compiling source code. This helps students understand the difference between compile-time and run-time errors. However, once students understand this difference, it is more efficient to let Eclipse perform the compilation on save. They can turn this on in Gild as it is a feature in Gild, but some other features (such as Code wizards) are not available options in the Gild editor.

Some students move onto Eclipse IDE, but are not fully ready for all its features. The Eclipse debugger can be quite complicated for a novice. We have found that some students choose to use the Java Perspective from Eclipse for their coding, but return to Gild to use the simplified debugger. Similarly, some students prefer the simplicity of Textpad and will use that tool for coding. Textpad does not have a debugger, so students will use Gild in order to use this functionality.

Gild as a set of preferences

Gild was developed as a plug-in for Eclipse, stripping out and simplifying features in the IDE; however, many of the lessons learned through our experiences show that Eclipse users would benefit from a more streamlined and intuitive interface. Our conclusion from this work is that many of the features in Gild that involve removing functionality should ideally be a set of preferences in Eclipse. New users of Eclipse, whether they are new to programming or not, still need to install, use and learn how to efficiently and effectively use the IDE.

An open source project, dedicated to supporting novice programmers, could provide a repository of plugins to provide additional pedagogical support for use in the classroom by instructors and learners. Some of the plugins we developed for Gild, such as the integrated web browser, integrated powerpoint support and additional support for compiler errors, could be initially added to this repository of plugins. There are also numerous other groups building educational plugins for Eclipse. A more ambitious plan would be to house course content (see the Gild units on the Gild Wiki) – this was the intent of the Ecesis project which was never fully realized.

Finally, there are many other features and plugins for Eclipse that could be very useful for educational purposes. For example, one of our projects, TagSEA (see http://tagsea.sourceforge.net/), was in part inspired from our work with Gild as instructors need a more powerful mechanism for relating code resources to course concepts and to facilitate student navigation of source code and documentation. Other useful tools include collaborative support and quality testing tools.