# Gild Feature Report

## *1 Introduction*

The purpose of Gild has always been twofold: 1) to introduce novice programmers to a professional IDE environment; 2) to support teachers with tools that will help students learn programming skills.

There are several high-level requirements that have to be fulfilled in order to accomplish (1)

A. **The environment must have the real feel of a professional IDE**. In real-world development, programmers use advanced tools to support their programming tasks. An introduction to programming environments should have the feel of a professional IDE in order to lessen the impact of moving to a professional environment.
B. **The environment must grow with the user**. As the novice becomes more skilled, the more powerful features of the IDE will become more important and useful. It is also important to introduce these features to the programmer so that he/she can become more knowledgeable about the tool, as well as about programming in general.
C. **The environment must support the user as he/she learns programming skills**. One of the most important functions of an IDE is that it helps programmers be more effective at their programming tasks. This is often achieved by assuming that the programmer is knowledgeable about programming in the first place. Functionality is added to the interface, and information is shown in the interface, according to this criteria. However, this can be a hindrance to novice programmers. Some details that are obvious to experts must be made explicit to novices, and some information that is useful for experts must be hidden for novices.

Section 2 of this document will discuss how some of the features of Gild attempted to address these high-level requirements, and what difficulties were encountered in the implementation of those features.

This document is mostly concerned with outlining the requirements and features for supporting novices in their programming tasks. Therefore, it will not discuss in detail the features and requirements that Gild had as a tool to support teachers

## *2 Novice Programmer Requirements*

In this section we will discuss some of the key features of Gild and how they addressed some of the high-level requirements that were outlined in the introduction of this

document. All of the high-level requirements had to be taken into consideration in the development of these features. So, we will discuss Gild on a feature-by feature basis and describe how the features relate to the high-level requirements.

## 2.1 The Gild Java Editor

One of the most important features of a professional IDE is the set of editors that it has for supporting code development. Gild was focused on helping novices understand programming concepts using the Java programming language. Therefore, a robust Java editor was developed. It was very much like the standard JDT Java Editor for Eclipse, but also different in several ways. The following table shows a comparison between important features the JDT Java Editor and the Eclipse Java editor.

| **JDT Editor** | **Gild Editor** |
|---|---|
| Syntax Highlighting | Syntax Highlighting |
| Line Numbers | Line Numbers |
| Inline errors, warnings, and markers | Inline errors, warnings, and markers |
| Bracket matching | Bracket Matching |
| Java Hot-spot code replace | Read-only editor while the JVM is running. |
| Code completion | |
| Quick-Fix helps | |
| Source Attachments for class files | |

One very significant difference between the JDT editor is the number of features that seem to be missing from the Gild Editor. These were purposely left out of the Gild editor. It was believed that the advanced features of code completion, and quick-fix helps, while very useful for advanced programmers, actually hinder novices in the basics of learning a programming language. Novices often need repetition in order to learn key programming concepts. Removing code completion and quick-fixes encourages that repetition. These features should be added as novices become more familiar with the language and programming concepts.

Also, the Gild editor did not display source attachments for .class files. The reason for this is that it was feared that displaying non-editable source code in the Java editor may cause confusion and loss of context for novices. Instead, a message was displayed indicating the difference between source files and compiled Java.

The Gild editor also disallowed editing of source code during the time that a program was being run. The reason for this is twofold: it was deemed important that novices understand the difference between compiled code and interpreted code that can be changed on the fly. Also, during the time that Gild was being developed, the Hot-Spot code replacer often failed with confusing errors. It was deemed better to hide these errors from novices. So, instead of allowing source edits to occur while a launch was running, the Gild editor became read-only and was highlighted with a yellow background, indicating to novices that they should “yield” from editing the source files.

## 2.2 The Gild Debugger

Another very important feature of a professional IDE is the debugger. It is very helpful for code understanding: both for novices and for professionals. However, novices often have a difficult time simply understanding a single thread of execution, so the Eclipse multi-threaded debugger was replaced with a simpler debugger that followed a single thread of execution. Having only one thread of execution is not a problem for beginner programming problems as the normally only have one thread anyway.

Since there was no longer any need to display threads, the Gild debugger displayed the current stack thread instead. This helped to make more efficient use of screen real estate, and it also helped to keep the context of the running program evident to the novice.

The simplified debugger also reduced the number of operations that could be used to simple *run, pause, stop, step-over, and step-into*. The step-into operation was enhanced to filter the locations that it could step into to files within the workspace. This decision was made because the importance of a debugger for novices is that they understand their own code. Stepping into code from third-parties can cause loss of context. Again, the more advanced features such as multi-threading and more advanced actions should be added as the novice becomes more skilled and requires the use of these features.

## 2.3 Gild Resource Management

One of the things that novices can find most confusing about programming is the process of organizing source code. The Java notions of packages and classpaths can be especially frustrating to novices, especially when they have not yet begun to grasp simple programming tasks such as writing a “hello world” application.

Also, for most novice tasks, the concepts of packages and classpaths are not necessary. What is more important at this beginning stage is the fundamental concepts of source code and compiled, executable code. In order to support the learning of these basic concepts, Gild took the approach of exposing novices to project and file resources rather than packages. Package explorers of varying degrees of detail could be added as more advanced organization of projects became required by the difficulty of programming tasks and the experience of the programmer.

Still, for simple programming tasks even the concept of a project is more than is necessary. Eclipse’s early design, which required every resource to exist in a project, was a constant source of frustration for novices. So, Gild created a “dummy” project for quick editing of files that either did not exist in the workspace, or that the students wanted to create quickly without the need to organize their projects.

In other cases, however, projects were very useful. For example, projects could be used to contain assignments created by teachers. In order to support this, Gild allowed the exporting and importing of projects in compressed .zip files. This feature was created for Eclipse early in the 2.x stream before it had importing and exporting of projects as a standard feature.

## 2.4 Gild's Pedagogical Support

The Eclipse IDE is a rich environment with a lot of support for expert programmers. However, it may be the case that the support that it gives to programmers falls just short of what novices need in order to learn programming skills. Here, we outline some of the features that Gild has in order to support learning.

- **Enhanced Tooltips.** Eclipse gives a lot of visual cues for things such as errors and warnings. These cues are available in many places, such as the Resource Navigator and the Package Explorer. However, the cues are often not accompanied by textual information that explains what they mean. Novices unfamiliar with IDE's and concepts such as compiler errors can have difficulty interpreting these cues. So, Gild added extra tool-tip information such as, "This file has an error," or, "This file needs to be built."
- **Separation of the Edit/Build/Run Process.** It was determined that it is often useful to make novices aware of the difference between source code and compile/executable code. Therefore, Gild turned off the auto-build features by default. It also gave extra cues to inform the programmer about what part of the programming process he or she is in. For example, if a file has been edited, but not compiled, extra label decorators are displayed in order to inform the programmer that files still need to be compiled.
- **One-click Run**. Gild took a different approach to launching programs than Eclipse does by default. Rather than requiring the user to configure launches for applications and applets in the workspace, Gild automatically configured launches based on the current selection in the workbench. For example, if a .java file with a main() method is selected in the workbench, and application launch would be configured for the compiled version of that file, and it would be launched.
- **Extensive Step-By-Step Documentation**. The documentation for Gild is organized in such a manner as to teach users how to use each feature by using step-by-step examples of their usage. This was done in order to reduce the learning curve required in learning the new IDE.
- **Extra Error Help**. Expert programmers have enough experience to interpret and understand simple compiler errors. However, these errors can be cryptic to programmers who are encountering them for the first time. To alleviate the frustration caused by this, Gild included 51 extra help messages for errors. The Gild error view displays both the original error message and contains a link to extra help and examples about how to fix the error.

# *3 Implementation Notes*

The implementation of Gild presented many technical challenges. The most significant of these is the fact that, though Eclipse has an extensive and excellent plug-in mechanism, it is still very challenging to filter information and features out of the Eclipse environment. We often found that we had to re-implement many Eclipse features in order to fulfill our requirements. In doing so, we had to re-write a lot of code that had previously been written very well by the Eclipse team, but was internal to the platform or the JDT.

Sometimes, we resorted to using internal code, but the result is that Gild becomes quite fragile and requires a lot of maintenance.

With the advent of Eclipse 3.x and RCP, we considered creating Gild as an RCP application. However, it was feared that in doing so, we would loose the advantage of a smooth transition into the full-featured Eclipse IDE. So, the path of RCP was ruled-out.

Recent versions of Eclipse have greatly improved filtering features. If these features continue to be designed specifically with novices in mind, Eclipse may become a much more useful IDE for novices.

## *4 Further Information*

The Gild website has some useful information about the design and features of Gild, as well as research about Gild and novice programmers. The Gild website can be found at http://gild.cs.uvic.ca.

## *Appendix I: List of Feature Requirements for Gild*

1. Files:
    1. Simple import of classes
    2. Simple method for saving files and profiles
    3. One-way CVS
2. Editor:
    1. Syntax highlighting
    2. Line numbers
    3. Code completion
3. Additional learning support:
    1. Help reference with a list of typical errors
    2. Meaningful error messages from the compiler
    3. Code snippets, examples
    4. Animations of code execution
    5. Catch "newbie" mistakes
    6. Visualizations of classes and concepts
    7. Tutorials that can be added by the instructor
    8. Display the Java API
4. Annotation tool for markers
    4. List of common comments
    5. Provide feedback through IDE
    6. Marking schemas
    7. Assignment sorting (e.g. "need to look at again")
    8. Bookkeeping functionality
5. Collaborative features
    4. Forums, newsgroups
    5. Messenger
    6. Allow instructor and students to see the same code
6. Features can be added to the environment over time (by instructors and students)
7. Separate save/compile/run
8. Simple debugger

Key:

Implemented
Unimplemented