Developing Marking Support within Eclipse

Del Myers, Elizabeth Hargreaves, Jody Ryall, Suzanne Thompson,
Marilyn Burgess, Daniel German, Margaret-Anne Storey
University of Victoria, BC, Canada

ABSTRACT

In this paper, we describe marking features provided in Gild, a set of plug-ins to support education in Eclipse developed at the University of Victoria. We discuss our requirements gathering techniques, design process and the challenges experienced during development of this tool. We also consider the problematic nature of student evaluation, particularly within the context of introductory Computer Science courses.

Keywords

Marking support, student evaluation, novice programmers, integrated development environments.

1. INTRODUCTION

Gild is a tool developed at the University of Victoria that is designed specifically to improve the teaching and learning of Java programming [1]. Built as a set of plug-ins for Eclipse, Gild currently offers a customized perspective of this IDE for students, as well as support for instructors using this tool in the classroom. Our tool includes a set of unique features such as an integrated web browser, a simplified debugger, and an enhanced resource view. Since initial deployments of the student and instructor views have been highly successful, we are now developing a marking prototype to provide assistance for evaluation of student assignments.

Gild has been used in introductory programming courses at the University of Victoria since January 2004. Approximately 200 students have been exposed to Gild through programming exercises, lab sessions and course lectures. There is continued interest in using Gild for teaching at the University of Victoria. In addition, interest in Gild has been expressed by other colleges, universities and research groups. Since the deployment of Gild, we have utilized several methods to gather feedback from students, including surveys, interviews and an ethnographic study. Thus far, feedback from students has been positive. Our studies have brought forward several usability issues that we have been able to address in subsequent versions. These issues include the lack of a bracket matching utility and requests for a simpler project and file creation process. Students have shown enthusiasm for the simplified debugger, project navigation and organization, and the recently improved project/file creation functionality.

Communication between students and instructors forms an important part of the educational experience. Our requirements for Gild include the development of a marking view to improve the feedback given to students and the efficiency of marking assignments. Our current prototype of the marking view provides valuable information about the ways teaching assistants want to use a marking tool in terms of the user interface and features available.

2. MOTIVATION

In order to understand why developing marking functionality is critical for Gild, we consider the context in which marking is performed and the benefits that integrated marking support provides for both instructors and students. Requirements gathered at the University of Victoria and Dalhousie University revealed that, invariably, marking must be completed quickly and accurately. Concerns about time constraints consistently emerge when we have reviewed iterations of the marking prototype with lab instructors and teaching assistants.

We elicited requirements for Gild through focus groups, design meetings, various conferences, and informal meetings with interested parties (e.g., teachers and teaching assistants). Our results showed that there was a definite interest in integrating features to provide feedback for students. This interest came mainly from teaching assistants who wanted tool support for their marking tasks. At the University of Victoria, teaching assistants are often assigned hundreds of assignments to mark each se-

mester; further to this, the University of Victoria allocates only 20 minutes of a teaching assistant's time per student per course. If they are well designed, marking tools could significantly increase the efficiency and effectiveness of the marking process. When incorporating elicited requirements into Gild, we considered Chickering and Gamson's seven principles for good practice in undergraduate education [2] as well as Walvoord and Anderson's twelve principles for managing the grading process [3].

Providing high quality feedback is a challenge within most university environments, particularly when teaching introductory programming courses with large class sizes. Classroom instructors are able to capture teachable moments by manipulating source code on-the-fly within Gild. Using marking support within Gild may potentially allow instructors to identify teachable moments for individual students, provided through comments given on student assignments. The relative simplicity of passing Gild projects between students and instructors provides an opportunity for rich feedback and improves the quality of instruction. The advantage of having a single application which addresses all of the instruction, evaluation and learning needs of instructors, teaching assistants and students is paramount.

3. GILD MARKING FEATURES

We began development of the Gild marking functionality by implementing an interface which allowed graders to annotate students' text and source code files with special "marks". The marks associated scores and comments with files in the workspace. Scores could easily be tallied and then persisted in XML to be communicated back to the student. The annotations could be specified ahead of time or added during the marking process. The marks could then be dragged from a specialized view onto the text like a quick way of writing comments onto a piece of paper [Figure 1].

It was found that although there is an advantage in having the marking features integrated with the IDE environment, the annotation of text on the screen was not significantly more efficient than simply printing the assignments and writing on them. What really interested the teaching assistants was an integrated method of automated marking in combination with the marking features. The majority of this paper will deal with the automated marking features of Gild.

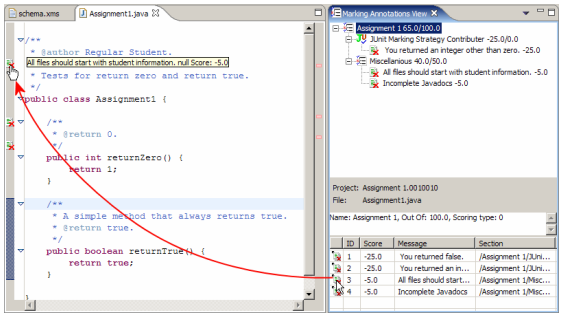

Figure 1: The drag-and-and drop interface for annotating assignments.

Programming courses in Computer Science lend themselves to automated marking. The halting problem shows that it is impossible to prove the correctness of any program via any automated means [4]. However, given the simplicity of first year programming assignments it is possible to run a program to check for expected output or for runtime errors such as uncaught exceptions. This kind of automated marking may greatly increase a grader's efficiency. The human grader can then review the marks assigned by the automated system, and potentially invest more time in giving constructive comments to the student.

3.1 Design Overview (User's View)

In our informal interviews with instructors and teaching assistants, we found that there exists a range of grading methods. We wanted to avoid any assumptions about how an individual or group would grade their assignments. This led us to base our marking features on the concept of a "marking schema". The schema is used as a standard method for grading multiple assignments. We assume that one project in the workspace constitutes one assignment. This is to simplify the calculation of grades. In Eclipse, a project normally signifies a comprehensive unit of work, so the mapping of projects to assignments is convenient. A marking schema is abstractly conceived as a tree structure that has "sections" as internal nodes and "notes" as leaf nodes [Figure 2].

Sections define the grader's conceptualization of how marks for an assignment are to be broken down. Sections have a name and an "out-of" or "maximum" score associated with them. Notes are the elements that will be used to calculate the grade of the assignment. Scores can be associated with

each note to facilitate the calculation of grades. Also, messages can be incorporated to provide feedback to the students. Notes from the schema will eventually be applied to individual files in an assignment.

```
[Section] Loop Assignment (Out of 100)
  [Section] For Loops (Out of 25)
    [Note] Wrong number of iterations.
        (Score -10)
  [Section] While Loops (Out of 25)
    [Note] Never terminates.
        (Score -20)
  [Section] Do Loops (Out of 25)
    [Note] Should not execute first
        time. (Score -10)
  [Section] Miscellaneous (Out of 25)
    [Note] One-off error (Score -5)
```

Figure 2: An example of a schema that could be used for marking a student's understanding of loops in Java.

The marking schema allows the grader to use one of two marking styles: 1) credit; or 2) demerit. Credit assumes that the student begins with 0 marks and is awarded marks until reaching (or possibly exceeding) the maximum score. Demerit assumes that the student begins with full marks and loses points for mistakes. Demerit also allows for "bonus" marks for example when a student shows an exemplary understanding of the material.

Gild includes a schema editor which is used to adjust all of the elements of a schema, and to associate projects with the schema that will be used to mark them [Figure 3].

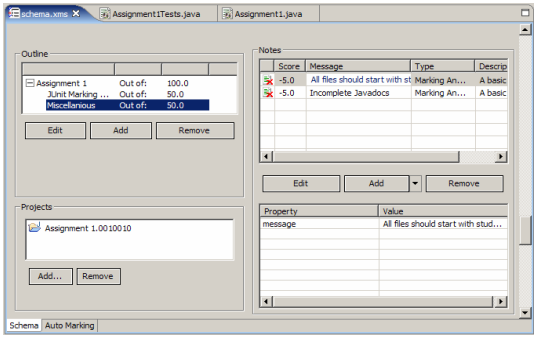

Figure 3: The main page of the marking schema editor.

3.2 Technical Overview

The marking schema is the foundation for implementing Gild's marking features. The basic tree structure is implemented using the Eclipse Modeling Framework (EMF). EMF allowed us to design the model in an interactive way without having to write a great deal of custom Java code for XML serialization, tracking containment, etc. To ensure all information that is relevant to the marking schema, (e.g., JUnit tests or scripts) is kept as a cohesive unit, we enforce that a marking schema be contained within a special "Marking Schema Project." The schema is stored in a file called schema.xms and is to the Marking Schema Project as the plugin.xml file is to Plug-in Projects in Eclipse. A persistent property in an assignment's project is set to associate the assignment with the Marking Schema Project that will be used for marking it.

From a small sample of automated marking tools that have already been created by professors at the University of Victoria, it was clear that the implementation of our tools had to be extensible. Eclipse offers a richly extensible platform. We took advantage of this in the design of our marking features.

The Gild marking features are based upon several custom extension points [Figure 4]. This implies that to extend the marking features, some Java programming is required. However, we feel that the added complexity is a worthy trade-off for extra flexibility. The model for the marking schema is implemented in the "Marking Model" plug-in. The "Marking" plug-in contains the core functionality for the Gild marking features. The "Marking Instructor" plug-in contains tools and functionality that will be of most use to course instructors and their assistants. For example, the marking schema editor is contained in the Marking Instructor plug-in.

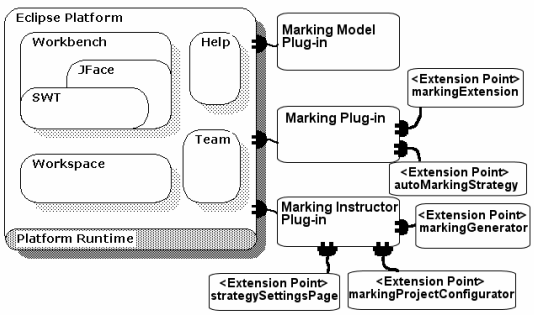

Figure 4: The Gild marking features' plug-in structure.

A thorough examination of each extension point is beyond the scope of this paper. Hopefully the example of § 3.3 will suffice to illustrate the usage of each extension point. At this point, we would like to discuss only two extension points.

The extension point markingExtension was implemented to allow flexibility in the marking schema data structure. It implements the idea of the "note" described above. The extension point makes it possible to define custom attributes for each note. For example, we use this extension point to contribute a basic annotation node that has only the attributes "message" and "score".

The autoMarkingStrategy extension point is the most important extension point from the perspective of automated marking. It is used to define what we call a "marking strategy." Marking strategies are used in two ways: 1) they provide the functionality required to automatically mark assignments; and 2) they define any extra data that is needed to perform that functionality. The extra data is persisted by extending the idea of a "section" that is defined in Section 3.1. An example marking strategy could use a Perl script to mark students' assignments. That marking strategy would be responsible for 1) invoking a Perl script to test the assignments; 2) choosing the notes that will be applied to the assignments as the Perl script is run; and 3) storing any extra settings that are needed by (1) (e.g., the file name of the Perl script).

After the marking schema has been defined, grading is relatively simple. As grades (i.e. "notes") are applied to assignments, another data structure, specific to the assignment, is populated with those grades. The data structure is much like that used for the marking schema except that there is no need for a marking strategy. Also, a note can be stored zero or many times for an assignment whereas it can only be stored once for a schema. The data for the assignment is persisted as an XML file. The file can be viewed by the student in specialized views in Eclipse, or it can be automatically translated into HTML.

3.3 An Example (Java and JUnit)

We decided to use JUnit in our prototype of automated marking for two reasons:

1) At the University of Victoria, Java is used to teach the concepts of computer programming. JUnit offers robust tools for testing whether the components of Java programs run as expected.
2) Eclipse comes bundled with the JUnit framework. Using JUnit in Gild means that students and instructors do not have to install additional software.

Exhaustive implementation details are beyond the scope of this paper therefore an illustrative example will be given instead with some implementation details provided.

Assume that an instructor for a course in Java has used Gild to create an assignment that her students are to complete. The objective of the assignment is to ask students to implement some methods. The instructor begins by creating a number of Java classes with the public methods that the students are expected to fill in.

At this point, the instructor can define the marking schema that is to be used. She selects "File > New Project" to create a new Marking Schema Project. A wizard is presented to her that asks her to enter a name for the project and select the projects that are to be marked using this schema. Right now, there is only one: the example project that she is going to give to her students. She selects this project and then selects "Finish".

The JUnit Automated Marking plug-in has made use of the markingProjectConfigurator extension point, so a configurator is invoked. The configurator tries to automatically generate skeleton code for JUnit tests. It does so by scanning the source code in the projects that were selected in the wizard. For every public class, a corresponding test class is generated with all the needed test methods.

With the JUnit tests created, the instructor can write the actual test code for each method. Each method will have JavaDoc comments, including two tags of interest: @score and @message. For each test that fails, the instructor wants one point to be deducted, so she types a -1 beside each @score comment. She also changes the @message comments to be appropriate for each test.

Now the instructor can edit the marking schema by opening it in the Marking Schema Editor. First, she would like to verify that all of the information for the JUnit testing is present in the schema, so she opens up the "Auto Marking" page of the editor. The JUnit Automated Marking plug-in makes use of the strategySettingsPage extension point, which presents the instructor with some settings information for the JUnit marking strategy. On this page, she can

select the test cases that she wants to use in automated marking. She selects all of them.

The JUnit Automated Marking plug-in also makes use of the markingStrategyContributer extension point, so a button saying "Contribute To Schema" is available. This button causes the JUnit plug-in to scan the selected JUnit tests for the JavaDocs, extracting the information entered in the @score and @message tags. For each test method, a specialized JUnit note (contributed using the markingExtension extension point) is added to the marking schema. When the instructor goes back to the "Schema" page of the Marking Schema Editor, she will notice that a new section called "JUnit Marking Strategy" has been added, with a number of notes. Satisfied by the schema, the instructor saves it. She then uses Gild's export features to archive the marking schema project and sends it to her teaching assistants. She also archives the assignment project to publish on her website for students to download and complete.

Once all of the students have finished their assignments and have handed them in, it is easy for the teaching assistants to automatically mark them. Each teaching assistant simply has to import the assignments assigned to her using Gild's import features. Every assignment will already be associated with the schema supplied by the instructor because the necessary information was persisted with the assignment before it was given to the students. The teaching assistant can still double-check that all of the right associations have been made by using the Marking Schema Editor. To mark the assignments, the teaching assistant simply has to open the "Auto Marking" page of the Marking Schema editor, and select the "Mark All" button. Since the JUnit Automated Marking plug-in is installed on the teaching assistant's computer, the JUnit Automated Marking Strategy (contributed using the autoMarkingStrategy extension point) is invoked. It uses reflection to run the methods in the supplied test cases, and applies marks to each assignment based on whether those tests succeed or fail. When the marking is completed, a file called "marks.mx" is created for each assignment. These files can be sent back to the students for viewing in Gild, or they can be converted to HTML documents for publishing on the web.

4. CONCLUSIONS/FUTURE WORK

Our objective is to integrate, within Gild, a marking environment that not only does automated marking, but that also supports the entire marking process: assignment collection, marking schema creation, automatic testing, annotation assistance, returning results to students, and tracking grades. In addition, we seek to provide a suite of tools that assists students, teachers and teaching assistants. To our knowledge there is no tool that supports these tasks for all stakeholder groups in a single integrated environment.

There is often insufficient support (at organizational and technical levels) for the marking aspect of instruction; each instructor has her own unique approach. Ease of use and efficiency is critical, and we anticipate that the development of Gild's marking view will be an iterative process. We plan to continue testing our prototype and deploy these features in January 2005. Please note that the latest updates of Gild are regularly made available at the Gild website [1].

ACKNOWLEDGMENTS

We would like to thank IBM CAS, Mary Sanseverino and the CHISEL group for their assistance with this project. We also thank our participants for their invaluable feedback.

AUTHORS

D. Myers, E. Hargreaves, J. Ryall, S. Thompson, and M. Burgess are Computer Science students and research assistants at the University of Victoria. M.- Storey and D. German are faculty members in the Department of Computer Science at the University of Victoria.

REFERENCES

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

[2] Chickering, A.W. and Gamson, Z.F. Seven Principles for Good Practice in Undergraduate Education. AAHE Bulletin, pp. 3-7, Mar 1987.

[3] Walvoord, B.E. & Johnson Anderson, V. (1998). Effective Grading: A Tool for Learning and Assessment. San Francisco: Jossey-Bass Inc. (pp. 9 – 16).

[4] Turing, A.. On computable numbers, with an application to the Entscheidungsproblem. Proceedings of the London Mathematical Society, Series 2, 42. 1936. (pp 230-265).