JRefleX: Towards Supporting Small Student Software Teams

Kenny Wong, Warren Blanchet, Ying Liu, Curtis Schofield, Eleni Stroulia, Zhenchang Xing
Department of Computing Science
University of Alberta
{kenw,blanchet,yingl,schofiel,stroulia,xing}@cs.ualberta.ca

Abstract

The success of a software development project depends on the technical competency of the development team, the quality of the tools they use, and the project-management decisions they make during the software lifecycle. Instructors of software-engineering courses that involve small project teams are often overwhelmed with the task of monitoring the progress of multiple teams. Without adequate monitoring and advice on best practices, problems in the team's process or the developed product may go unnoticed until it is too late to be easily fixed. This paper introduces the JRefleX environment, with components built upon Eclipse, to support the education of small software teams.

Keywords: software process, small teams, project management, software quality, software engineering education

1. Introduction

Changing requirements, tight delivery schedules, and developer turnaround are common challenges facing almost every software development team today. To address these challenges effectively, developers must have a good understanding of their project status, have sufficient programming experience, collaborate well within their teams, and adapt efficiently as needs dictate.

Such abilities are difficult to teach and acquire in a university software-engineering course. Instructors are eager to teach their students with industrially relevant tools and skills, but closely monitoring and mentoring a large number of small software teams in a course is a major issue. In our experience, involving roughly 30 small teams in a course of over 120 students, there are often major variations among the team projects and the skills of team members, making the detection of individual problems too subtle. Students may get mired in the complexity of the product or their individual components, and not recognize signs of problems in their overall design or development process early enough to effectively involve the instructor.

This year, we embarked on the JRefleX project, whose goal is to develop a tool to monitor the collaboration process of software teams and to aid the understanding of changes in software designs. In particular, we are aiming to infer high-level information about how teams work together, including team-organization style and the impact of each member's contribution or lack thereof. The data to support these needs is gathered unobtrusively as developers work on their code. Visualizations are created and delivered to the instructor so that he can see an up-to-date view of their progress and make comparisons across teams. With a coherent view of team collaboration and deliverables produced, we expect the instructor to better manage these projects and provide relevant, timely, and informative feedback. Certain views are valuable enough that we are considering making them available to the students to monitor themselves and to see how their teams might rank against others in the course. We hypothesize that teams who are aware of their own collaborative process, reflect upon their progress, and make adjustments as needed are more likely to make the right project-management decisions when new challenges arrive.

In the longer term, the JRefleX environment is to provide an experience repository for the collaborative development processes of a series of projects. Such a repository could be data mined to discover interesting correlations between objectively collected process and product data, subjective developer perceptions of their own work, and their performance as assessed in their project marks.

The JRefleX environment currently uses CVS (Concurrent Versioning System) to support collaborative software development. The environment accesses the CVS history of a team's software development work on a project. The analysis modules are implemented as Eclipse plugins. As well, JRefleX provides a Wiki-based user interface to deliver the results of analyses in a web-based format.

Section 2 introduces the architecture of JRefleX. Section 3 briefly describes the database that manages the process and product metadata storage for the environment. Section 4 outlines an Eclipse plugin to provide analyses of the team's collaborative process. Section 5 outlines an Eclipse plugin to show analyses of software design changes. Section 6 briefly summarizes our early experiences.

2. Architecture

The JRefleX environment consists of three main parts:

- the repository,
- the analysis components, and
- the development environment (based on Eclipse and a web browser).

2.1. Repository

The repository consists of:

- CVS, where all development work products are actually stored;
- a database, where work product metadata and analysis information about the software process and its products are maintained; and
- a Wiki server that delivers web-based reports to students and instructors using the database content.

2.2. Analysis components

The analysis components are responsible for analyzing the database and CVS content to discover information about how a team has collaborated, as captured in the history of member actions and software changes. We have been developing a set of heuristics for understanding the way team members work together. In particular, if a team member has authored, and has regularly modified most of the code files in his team's CVS area, especially if this member began making changes earlier than the others, we might infer that this member is the "team leader". Alternatively, if a team member has made small changes to a variety of code files authored originally by other members, we might infer that this member has a "debugger" role on the team. Finally, if each team member has authored and maintained a specific set of code files with no changes to these by another member, we might infer that the team has decomposed the application into a number of fairly independent components, each assigned to a team member. This level of understanding and insight is very valuable for instructors (and managers) to follow the progress of a project effectively. Combined with cross-team summary reports delivered by the Wiki server, instructors can compare the performance for multiple teams.

2.3. Development environment

The development environment is based partly on Eclipse and interoperates with CVS. One eventual purpose (besides supporting software construction) is to record unobtrusively the fine-grained tool actions of developers working upon their code and documentation. Also, Eclipse provides the platform for which plugins can visually present the analysis results to instructors or students (without requiring a web browser).

In parallel to the Eclipse tool, the development environment provides a set of browser-accessible services, including TSP/PSP [1] forms where developers self-assess and provide information about their development process, to be stored also in the repository. This information includes estimates of planned work and peer reviews of individual contributions to various development activities. The services also include reports, generated by the Wiki server, which summarize and visualize the various analyses of software process and products.

The architecture of JRefleX depends on Eclipse as the main development tool to provide a seamless integration of software construction and analysis reporting activities. However, teams that do not adopt Eclipse for development can still gain much of the benefits of JRefleX as long as they use a web browser and CVS. (Due to the database layer, JRefleX is actually largely independent of CVS.) This is a practical adoption issue since Eclipse is computationally intensive (especially on the students' home computers), and our transitions to it in the lab environment must be done in phases.

3. Database

This section describes the JRefleX database core which stores work product metadata, analysis information, and self-assessment information.

The database core provides the underlying structure which includes the following basic concepts: CourseTerm, Project, Team, Member, WorkProduct, Version, and in the future, Activity and Quality.

In particular, a CourseTerm represents a particular group of Projects that are being developed for a class project in a specific academic term. A Project represents a particular module or portion of a module within a CVS area, and it is associated with a specific Team. A Team is a group of Members who are working together on one or more Projects.

Projects, Teams, and Members lay the groundwork for a particular piece of a Project, referred to as a WorkProduct. A WorkProduct represents a file within the Project's CVS area, and is anything requiring constructive effort by a specific Member. The actual information regarding what a Member has produced is stored as a Version of a WorkProduct. A Version parallels the notion of a CVS file revision and contains much the same metadata.

Future concepts include Activity and Quality. An Activity describes a particular type of work that Members may do while working on Projects. Such activities include planning, design, coding, testing, documentation, etc. This functionality would require instrumentation enhancements to Eclipse. A Quality describes a particular kind of non-functional requirement that is of interest for a Project, which instructors use for product evaluation. These qualities include learnability, usability, and extensibility.

There are other details to the database, including administration, population through CVS events, assessment form construction, and Wiki support, but for brevity they have been omitted here.

4. Collaboration Analysis

This section outlines the collaboration analysis component, which provides insight and understanding into the way teams work together, thus potentially alerting instructors and the teams themselves of potential problems.

This component consists of several parts, including:

- collaboration Wiki report, which presents collaboration analysis diagrams;
- collaboration Eclipse plugin, which conveys these views for Eclipse users (see Figure 1); and
- an analysis daemon, to recalculate collaboration analyses, software metrics, and development statistics daily (except data mining calculations which are done weekly).

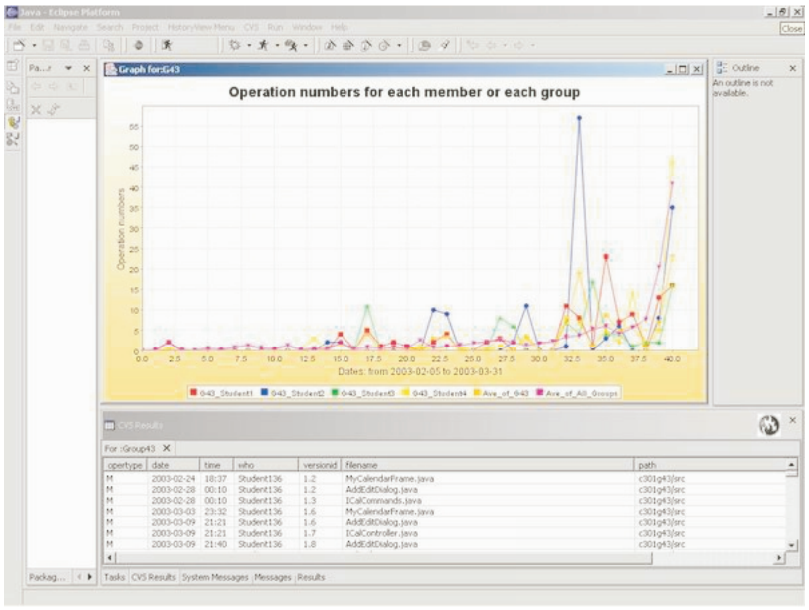
The results of these analyses are currently shown only to instructors and teaching assistants. We are currently exploring data mining techniques to extract valuable, but less obvious information from the accumulated datasets. For example, suppose file f1 is always modified by student s1 once the file f2 is changed by student s2 (implying some relationship between the two files). If the analysis finds that file f2 is changed once again, but file f1 has not been modified for some time, a reminder should be sent to student s1. (Such reminders and mentoring possibilities are the topic of future work.) Additional analyses are described in [2].

5. Evolution Analysis

This section describes the evolution analysis component, which enables developers and instructors to understand how a design has changed over time. The goal of this component, centered upon an Eclipse plugin, is to reveal the evolution style of a software system, the change profile of individual classes, co-dependent changes, and occurrences of refactoring.

5.1. Approach

The main input for the plugin is a sequence of design models, represented in XMI, corresponding to a sequence of snapshots of an object-oriented


Java - Eclipse Platform
File Edit Navigate Search Project HistoryView Menu CVS Run Window Help
Graph for:G43
Operation numbers for each member or each group
Operation numbers
0
5
10
15
20
25
30
35
40
45
50
55
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0
27.5
30.0
32.5
35.0
37.5
40.0
Dates: from 2003-02-05 to 2003-03-31
🟥 G43_Student1 🟦 G43_Student2 🟩 G43_Student3 🟨 G43_Student4 🟨 Ave_of_G43 🟪 Ave_of_All_Groups
Outline
An outline is not available.
CVS Results
For :Group43
Tasks CVS Results System Messages Messages Results

opertype	date	time	who	versionid	filename	path
M	2003-02-24	18:37	Student136	1.2	MyCalendarFrame.java	c301g43/src
M	2003-02-28	00:10	Student136	1.2	AddEditDialog.java	c301g43/src
M	2003-02-28	00:10	Student136	1.3	ICalCommands.java	c301g43/src
M	2003-03-03	23:32	Student136	1.6	MyCalendarFrame.java	c301g43/src
M	2003-03-09	21:21	Student136	1.6	AddEditDialog.java	c301g43/src
M	2003-03-09	21:21	Student136	1.7	ICalController.java	c301g43/src
M	2003-03-09	21:40	Student136	1.8	AddEditDialog.java	c301g43/src

Figure 1. Count of student actions over time

application, generated by regular checkouts from the CVS area for a project. XMI models can be reverse engineered from the application code, using roundtrip engineering tools such as Borland Together and Rational Rose. The core of the plugin relies on recovering the structural design changes from one version to the next. That is, the plugin implements a UML differencing algorithm that can extract surface changes to classes and interfaces, attributes, methods, and inheritance relations in terms of element additions, deletions, moves, and renamings. The algorithm produces change trees that report the deltas of the compared versions.

Aggregate information can then be extracted from a sequence of such change trees. By examining and analyzing the aggregate data, we can obtain a quick overview of the whole application evolution history. In particular, we can recover the overall software evolution history at three different levels:

- at the system level, we can identify different evolution phases and styles through the application evolution history;

- at the class level, we can recognize different classes according to their change history, such as continuously modified classes versus legacy classes;

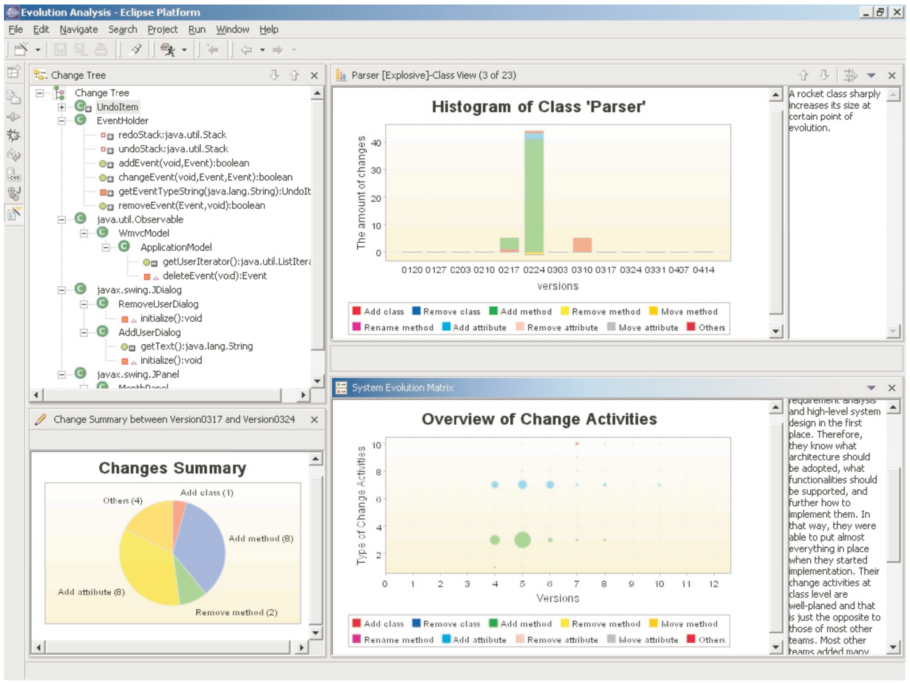
- at the change-tree level, we can identify various change patterns, such as refactorings.

5.2. Implementation

The Eclipse evolution analysis plugin works incrementally. Given a sequence of XMI models, the plugin reads the XMI files and parses them into class hierarchy trees, then runs the tree differencing algorithm against these trees, and saves the deltas into change trees. The change trees are analyzed further to present the following perspective, which contains four main views (see Figure 2).

The system matrix view shows a matrix that provides a quick overview to understanding the overall evolution history of the software project. Each column represents a version of the software, while each row represents different types of changes shown in different colors. The area of the bubble represents the number of each type of change. Thus, a bubble of size s at the (x,y) point in the matrix shows that s changes of type y happened between version x-1 and x. The exact number for s can be obtained from change summary view.

The class view depicts the change profile of each individual class. The changes can be shown in two different diagrams, matrix or histogram. The class matrix is similar to the system matrix, but it shows the changes of a particular class instead of



```
Change Tree
  UndoItem
  EventHolder
    redoStack:java.util.Stack
    undoStack:java.util.Stack
    addEvent(void,Event):boolean
    changeEvent(void,Event,Event):boolean
    getEventTypeString(java.lang.String):UndoIt
    removeEvent(Event,void):boolean
  java.util.Observable
    WmvcModel
      ApplicationModel
        getUserIterator():java.util.ListItera
        deleteEvent(void):Event
  javax.swing.JDialog
    RemoveUserDialog
      initialize():void
    AddUserDialog
      getText():java.lang.String
      initialize():void
  javax.swing.JPanel
```

A rocket class sharply increases its size at certain point of evolution.

and high-level system design in the first place. Therefore, they know what architecture should be adopted, what functionalities should be supported, and further how to implement them. In that way, they were able to put almost everything in place when they started implementation. Their change activities at class level are well-planed and that is just the opposite to those of most other teams. Most other

Figure 2. Evolution perspective views

the whole system. The histogram uses a stacked bar chart to display changes. It is easier to see the number of changes in the histogram, while the matrix is more convenient to find evolution styles. The plugin analyzes the change profiles of individual classes, and assigns them different evolution types, such as explosive, short-lived, idle, etc., which are shown in the view title. A query mechanism is implemented to filter out the unwanted evolution types and allow the users focus only on types of interest.

The change tree and change summary views show detailed and summarized information of changes between two consecutive versions. The change summary view shows a pie chart that summarizes the number of different types of changes, while the change tree view presents the detailed changes to classes/interfaces and their attributes/methods in an explorer-style tree. The different icons represent class, interface, attribute, and method respectively. The different adornments on an icon represent different types of changes, such as plug sign for addition, empty triangle for changes like adding parameter, etc. Double-clicking a specific element will bring up the Java source editor.

6. Conclusion

Our work on JRefleX is still one in progress, and much research remains to properly test our hypothesis. Nevertheless, we have collected some promising experiences on how such an environment could be deployed in a third-year computing science software design course (using an earlier prototype). An incredible amount of data can be extracted, but the challenge for enabling understanding is to find the right combination of heuristics and views.

This work was supported by CSER, the Consortium for Software Engineering Research, and an IBM Eclipse Innovation Grant.

References

[1] Humphrey, W. “PSP/TSP”, <`http://www.sei.cmu.edu/tsp/watts-bio.html`>

[2] Liu, Y.; Stroulia, E. “Reverse Engineering the Process of Small Novice Software Teams” Working Conference on Reverse Engineering (WCRE 2003: Victoria, BC; November 13–16, 2003).