Applying PSP to Support Teaching of Programming Courses

Chien-Hung Liu[1]*, Shu-Ling Chen[2], and Chia-Jung Wu[1]

[1] Department of Computer Science and Information Engineering
National Taipei University of Technology
Taipei 106, Taiwan, ROC
{cliu, t5598022}@ntut.edu.tw

[2] Department of Management and Information Technology
Southern Taiwan University
Tainan 700, Taiwan, ROC
slchen@mail.stut.edu.tw

Received 20 June 2008; Accepted 16 July 2008

Abstract. The programming assignments are an essential means in programming courses to help students develop problem-solving skills and improve their understanding of programming concepts. By analyzing the process about how students practice their assignments, instructors can obtain valuable insights into student performances and understand their learning progresses. However, gathering the process data of assignment practices not only can be time-consuming and error-prone, but also can impose significant overheads to students. This paper describes how PSP can be useful in examining the process of students' programming practices and discusses several process statistics and their indications to student performances. A supporting tool is proposed to facilitate the tracking and analyzing of the process data. In particular, the proposed tool can automatically gather the process data for students, such as the size of programs, the time spent on the assignments, and the number of injected and removed defects. Moreover, based on the collected data, the tool can provide various statistical reports to facilitate the analysis of student performances and the understanding of students' programming problems so that instructors can develop a more effective teaching strategy to help students improve their programming skills.

Keywords: PSP, programming courses, programming assignments, process metrics

1 Introduction

In today's information age, software has become essential to our lives. In particular, software has been used to control or support different kinds of applications in all areas, such as business, transportation, and communication. Thus, many computer science departments offer various programming courses to enhance students' programming skills. In order to ensure that students have sufficient practices to develop problem-solving skills and improve their understanding of programming concepts, most instructors of programming courses will give students programming assignments (or laboratories). From the results of programming assignments, instructors can evaluate student performances and have an idea about students' learning progresses. Ideally, instructors can obtain valuable information about students' programming skills by examining whether the students' programs satisfy the assignment requirements. However, students may discuss or share their ideas for solving the assignments or even copy someone else's work. This could result in a situation that students seem to be able to complete their homework assignments by themselves, but their programming skills actually are not improved too much. Thus, instructors may misunderstand students' learning progresses if they rely only on checking whether the students' programs satisfy the assignment requirements.

Although many approaches [1, 2] have been proposed to detect the plagiarism in programming assignments and prevent students from copying other's programs, the detection of plagiarism does not provide additional information to instructors for understanding students' programming skills and learning experiences. To know students' learning statuses and to help students improve their programming ability, instructors need more information, such as the efforts spent on the assignments, the problems encountered during the assignment practices, and the quality of students' programs. By obtaining and analyzing the information about how students practice

* Correspondence author

their programming assignments, instructors can have a better understanding of students' learning processes and recognize students' problems in software development.

To obtain the information about the process of assignment practices, the approach of Personal Software Process (PSP) [3] can be very helpful. The PSP is proposed to help programmers improve their personal performance in software development. It defines several methods and practices that can be gradually applied in developing small programs. In particular, the PSP discipline requires programmers to gather their process data, such as program size, development time, and defect information. Through tracking and analyzing the process data, programmers can understand their talents in software development and determine which technologies to adopt or which methods work best for them. It has been shown that PSP can significantly reduce the number of defects, increase the software quality and programmer's productivity, and improve the predictability of software process [4, 5].

Based on the concepts of PSP, instructors can understand the learning experiences of students by tracking and analyzing students' process data of programming practices. For example, through gathering the size, time, and defect information of the practices, the student performances, such as productivity and defect density, can be derived and evaluated. By analyzing the trends of various process data, instructors can obtain valuable insights into the learning progresses of students so that they can develop a better teaching strategy and improve their teaching quality of programming courses. Meanwhile, the analysis of the process data also allows students to better understand their personal programming ability and recognize their problems in programming, and hence, provides an important foundation for students to make improvement.

Although the process data of students' assignment practices can be valuable for instructors to improve their teaching quality and for students to understand their programming ability, gathering the process data can be tedious and time-consuming. For each assignment, students have to gather various process data manually, such as the size of each class or program module, the time spent on each process phase, the number of injected and removed defects, the defect injection and removal phases, and defect fix time. This can impose significant overheads on students, especially for those students taking the introductory programming courses. To reduce these overheads, an automatic tool used to facilitate the data gathering becomes important so that students can concentrate on their programming practices without the distractions from process data gathering.

To support the teaching of programming courses, this paper adopts the concepts of PSP to gather and assess the process data of students' programming practices. Several process statistics are proposed and their indications to student performances are discussed. An automatic tool, called the Programming Data gathering and Analysis Tool (PDAT), is proposed to facilitate process data tracking and analysis. In particular, the proposed tool is based on the client-server architecture. The client of PDAT is developed as an Eclipse plug-in, where Eclipse is a widespread popular open source IDE [6]. With the PDAT client, the process data can be automatically gathered while students use Eclipse to develop their programming assignments. The collected data are then sent to the PDAT server automatically for data consolidation and analysis. From the analysis results, the PDAT server will provide several statistical reports that show the process statistics and trends for each individual student's performance and the overall performance of the entire class. Based on the performance results, instructors can design more appropriate teaching strategies and materials those can help students improve their programming skills more effectively.

The rest of the paper is organized as follows. In the next section, several considerations for applying the PSP to programming courses are described. Section 3 presents the approach used in the PDAT to gather the process data of assignment practices. Section 4 describes various process statistics and their possible indications. Section 5 presents the architecture design of the PDAT and illustrates its implementation. The last section summarizes the conclusions and describes our future work.

2 Integrating PSP Concepts into Programming Courses

Although the PSP can be taught as a stand-alone course [7, 8, 9], a number of studies have incorporated PSP concepts into programming courses [8-13]. Since the PSP involves various process practices and statistics, a complete PSP methodology is much too complex for those students who are beginning programmers. Thus, most programming courses that incorporate PSP introduce only parts of PSP concepts, such as the notions of process, planning, and estimation. Students are then asked to record their time spent on the programming laboratories and the defects found in the laboratories. The major purpose for introducing PSP concepts in programming course is to help students develop good programming habits in an early stage and to help novice programmers manage their time better.

As reported in [9, 12, 13], the integration of PSP into programming courses can have several advantages. For examples, the concepts of PSP can help students understand the software development process better and establish better concepts of planning, estimation, measurements, and data analysis. However, the incorporation of PSP into programming courses also has a number of problems. For instances, the data collection adds much overheads to students even with the assistance of supporting tools like spreadsheet [9]. Moreover, students have the tendency to record defects less accurately [8].

The incorporation of PSP into programming courses can help students understand the software process in the early stage and establish better ability in the size estimation and time management. However, students taking introductory programming courses are more concerned with programming than with the software process issues. Since our research purpose is to help instructors assess student performances and improve their teaching quality of programming courses, we attempts to minimize the PSP concepts to be taught in the programming courses in order to reduce students' overheads and avoid possible distractions from programming practices. Specifically, only basic concepts of PSP process and data collection forms, such as time log and defect log, are introduced so that students can understand the data gathered by the PDAT. Students are asked to use Eclipse with the PDAT plug-in to develop their assignment programs. The finished programs and the automatically gathered data are then submitted to the PDAT server for further analysis. As a result, students can get rid of unnecessary interruptions and overheads and are able to concentrate on the programming.

Moreover, in programming courses, the sizes of assignment programs are usually small and most of defects appeared in the programs can be either revealed by compiler or detected by students through program testing. Such defect information is important for instructors to understand the programming mistakes made by students. Thus, in order to gather the defect data revealed via program testing, students are asked to perform unit testing. Since Eclipse can be integrated with different unit testing tools, such as JUnit [14] or cppUnit [15], both the compilation and program testing defect information can be obtained through the Eclipse platform.

In addition, to gather complete and consistent defect data of program testing, it is suggested that instructors provide unit test cases for the assignment programs. Although unit testing is crucial to software development, most beginning programmers lack the skills of designing satisfactory test cases to test the programs adequately. Thus, instructors can define the interfaces (i.e., skeleton code) required for the assignment programs, such as the names and data types of the methods and parameters, and provide essential test cases based on the interfaces to students. In such a case, instructors can ensure if the programs are correctly implemented by students while obtaining more complete and consistent defect data.

3 Collection of the Process Data

Gathering process data manually can be time-consuming and error-prone. It will impose additional overheads on students and result in significant distractions while students are programming. In order to reduce students' overheads, the PDAT is designed to collect the process data automatically. Basically, this can be achieved by assessing the information of Eclipse workspace and plug-ins through their corresponding extension points [16]. Through the extension points, the events and data of program editor, compiler, and unit testing plug-ins can be obtained, and hence, the process data, such as the program size, the time spent on the program development, and the defects revealed by compiler or unit testing tool, can be collected automatically [17].

In order to facilitate the data gathering, the development process of assignments is divided into the *code* and *test* phases. In the code phase, students will focus on the program development and the time spent on the code phase includes the coding time and the fix time of compiler errors. Once the compilation is success and the unit testing is commenced, the process phase will change from the code phase to the test phase. In the test phase, students will focus on the unit testing and the time spent on the test phase includes the test execution time and the defect fix time. Note that the process defined in our approach is different from that of the PSP in which a process contains the plan, design, code, compile, test, and postmortem phases. This is because that, in our programming courses, Eclipse is used only for developing and testing programs. It will not be used for planning or designing the assignments. Moreover, the Eclipse supports continuous compilation in order to save the compilation time. This means that, when students are writing code, Eclipse will continuously compile the programs and show the syntax errors found so far. As a result, there is no specific compile phase as defined in the PSP process structure. Thus, the PDAT is designed to gather the process data only for the code and test phases.

According to the concepts of the PSP, three kinds of data including the size, time, and defect data need to be gathered. Table 1 lists the types of process data to be automatically gathered in different phases.

In Table 1, the lines of code (LOC) for each method, the number of methods for each class, and the number of classes for each assignment program are gathered during the code phase. The development time is the period that students employ Eclipse to write and compile the programs in the code phase. During this period, the program errors detected by compiler are considered as syntax defects and the error messages will be recorded for further analysis. Moreover, if there is any interruption to the tasks in the code or test phase, the interrupt time will not be counted when gathering the time data. For example, students may take a break, such as going to lunch, and tem porarily leave the programming tasks[1]. Such interrupt time will be ignored when computing the time data of the process.

Once students start to perform unit testing, the process of assignment practices then enters into the test phase. The process will remain at the test phase until all the test cases are passed. In the test phase, students need to test the program and remove all the defects identified by the test cases. As compared with the code phase in which the defect data are gathered from the compiler, the defect data of test phase are captured through the unit testing plug-in of Eclipse. Moreover, different types of defects can have different influences to students. For example, a logical error is usually more complex than a syntax error, and hence, is more difficult to remove. Thus, in order to understand the effects caused by different defects, the PDAT collects various defect data including the number of defects, the type of defects, the injection and removal phases, the fix time of defects, and the defect descriptions.

To obtain the defect fix time, the defects found by compiler and those found by unit testing tool are considered. Basically, the fix time will be the elapsed period between the time when defect error messages appear and disappear in the corresponding problem view of Eclipse or the failure window of unit testing tool[2]. If there are more than one defect disappeared in the period, then the fix time for each disappeared defect will be the average of the time period. Note that the defect fix time automatically obtained by the PDAT may not be fully accurate since there could be some interruptions during the programming process. To measure the precise time period of interruptions, it will require human interventions since there is no way to determine if students are thinking of the design or taking a break when they are not interacting with the computers. Since the size of the programming assignment is usually small, we will assume that there are only a few or no interruptions during the process. Thus, the defect fix time gathered by the PDAT can be close to the actual time spent on fixing the defects.

Table 1. The types of process data to be collected

Data/Phase	Code	Test
Size	numbers of class (NOC), numbers of method (NOM), lines of code (LOC) of each method	none
Time	development time	test execution time
Defect	number of defects (syntax), defect type, injection & removal phases, fix time, defect description	number of defects (functional), defect type, injection & removal phases, fix time, defect description

4 Analysis of the Process Data

After students complete their assignment programs and submit their process data to the PDAT server, instructors can analyze the student performances based on the gathered data. The process data can be analyzed from the perspective of an individual student or entire class. Several analysis reports regarding individual students and the entire class are discussed in this section.

4.1 Analysis of the Process Data for Individual Students

For analyzing the process data of individual students, several statistical reports that can facilitate the understanding of students' programming abilities and the improvement of teaching quality are considered. These process statistics include (1) time statistics; (2) familiarity of syntax; (3) defects hard to fix; (4) frequent defects; (5) productivity; and (6) defect density. The indications of the process statistics and how to obtain these statistics are discussed as follows.

- Time statistics

  The analysis of time data is concerned with the time that students spend on each phase and the time that students spend on each program assignment. By analyzing the time data, instructors can understand the efforts that students spend on the assignments and can evaluate if the loading of the assignments is appropriate to students. In addition, by taking into account the time statistics, productivity, and defect density, instructors

[1]We assume that students will save the programs or exit Eclipse when their programming tasks are interrupted.

[2]We assume that students will attempt to fix only one defect each time.

can know if students encounter difficulties when practicing the assignments. In general, when students spend too much time on the assignments and also have a low productivity and a high defect density, it implies that the students could encounter some problems in the program development. Instructors then can identify the possible causes of the problems through analyzing the defect data of the students.

In addition, the ratio of the time spent on the code phase and the time spent on the test phase can be used to evaluate if students carefully carry out the planning and design of the assignments. When students have extremely high ratio on testing time to development time and also have a large number of defects, it usually indicates that the students rush into coding without adequate planning and design. Thus, more defects could be injected into the programs, and hence, more testing time is required in order to remove the defects.

- Familiarity of syntax

  The familiarity of syntax is concerned with students' understandings to the programming language. Through analyzing the defect data gathered in the code phase, instructors can assess if students are familiar with the syntax of programming language. The defects found by compiler in the code phase are usually related to the programming syntax. For example, if students forget to declare an identifier, the compiler will generate an error message like "xxx cannot be resolved." If the students make typographical errors or miss semicomma, they may see a compiler error message similar to "yyy syntax error zzz." Thus, instructors can predefine some keywords that are related to the syntax errors in programming languages, such as "cannot be resolved." Then, by analyzing the frequency of such keywords appeared in the descriptions of compiler defects, instructors can presume the degree of students' familiarities to the syntax of programming language.

  In addition, instructors can analyze the descriptions of similar compiler error messages that frequently appear in student's historical defect data. From the causes of these syntax errors, instructors can know which programming syntax or concepts are misunderstood by students. For instance, if the frequent syntax error is associated with "null pointer," it usually suggests that the students do not have a clear concept about the pointer initialization. Moreover, a frequent error message associated with "type mismatch" may indicate that students do not understand the definitions of various data types. Thus, from the scope covered by the assignment practices and the gathered compiler defects, instructors can easily realize the students' programming syntax familiarities and help the students to clarify their misunderstandings.

- Defects hard to fix

  The defects hard to fix are concerned with the defects that have large fix time. These defects can be used to identify the possible problems encountered by students during their programming practices. Most of the time a complex defect will have a larger fix time than that of a trivial one. In general, the compiler defects are injected due to typo, omission, or unfamiliar with the programming syntax. Such defects usually will not take students too much time to fix or be the difficult problems to students. In contrast, the logical defects typically relate to the logic design of the programs. Such defects are usually complex, and hence, require more time to analyze and fix. Thus, instructors can know the problems that cause students much trouble by analyzing the defects hard to fix in their defect data.

  The defects hard to fix can be identified by sorting the fix time of the defects. The descriptions of these defects may provide the clues for students' problems. By considering the defects hard to fix and taking into account other process statistics, instructors can assess if students have difficulties in understanding the programming concepts or design algorithms required for solving the assignments. If most students have similar problems, this may indicate that instructors did not explain the programming concepts or design well enough. As a result, students need to spend more efforts in order to figure out how to fix the problems.

- Frequent defects

  The frequent defects are concerned with the occurrences of defects in students' historical defect data. If similar defects appear very often, it could suggest that students constantly make similar mistakes. By analyzing the descriptions and causes of such defects, instructors can know if students have clear concepts about the uses of programming constructs, data structures, or design algorithms. For example, students can have frequent defects like "assertion failed error," "array out of boundary," and "memory allocation error." Such defects can be identified by examining the error messages of defects based on some predefined keywords. These defects may suggest that the students need to enhance their understanding on the particular programming construct, data structure, or exception handling. Instructors should clarify the corresponding concepts again in order to help students not to produce similar defects repeatedly.

- Productivity

  The productivity is concerned with the program implementation efficiency of students. It can be used to partially assess student performances. For each assignment, the productivity of students is the ratio between the size of program and the time spent on the program. The average productivity will be the ratio between the total LOC (i.e., Total_LOC) of all assignments and the total time (i.e., Total_Hours) spent on the assignments. The formula for computing the average productivity is given below:

$$\text{Average Productivity} = \text{Total\_LOC} / \text{Total\_Hours}. \quad \textbf{(1)}$$

By considering the productivity of students and the quality of their programs, instructors can understand the programming talents of students [18]. In general, a student with high productivity and low defect density has a better programming ability. In addition, from the trend of the productivity, instructors can understand if the programming skills of students are improved. Moreover, by comparing personal productivity and defect density with class averages, students can have a better understanding for their own programming ability when compared with others. Figure 1 shows an example of student's productivity. The figure shows that the productivity of the student varies for different assignments. The trend of the productivity also indicates that the productivity of the student is gradually improved.

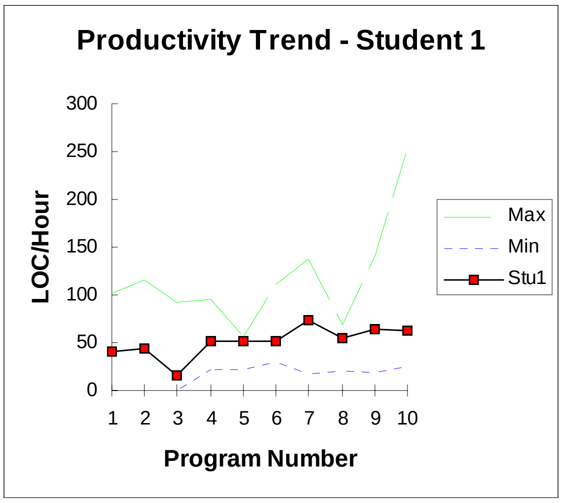


Fig. 1. An example of student's productivity trend

Note that the productivity may not accurately reflect students' programming talents. Students may rush into coding, and ignore the exception handling required for the assignments. As a result, the students may have a high productivity at the expense of poor software quality. Thus, instructors need to consider both the productivity and defect density in order to assess student performances. Moreover, based on students' productivity, instructors can estimate the possible effort required for students to complete the next assignment. The effort estimation can be valuable to instructors for adjusting the loading of the assignments.

- Defect density

The defect density is the number of defects found in per thousand lines of code (KLOC). It can be used as a metric to represent the software quality. The formula for computing the defect density is given as follows:

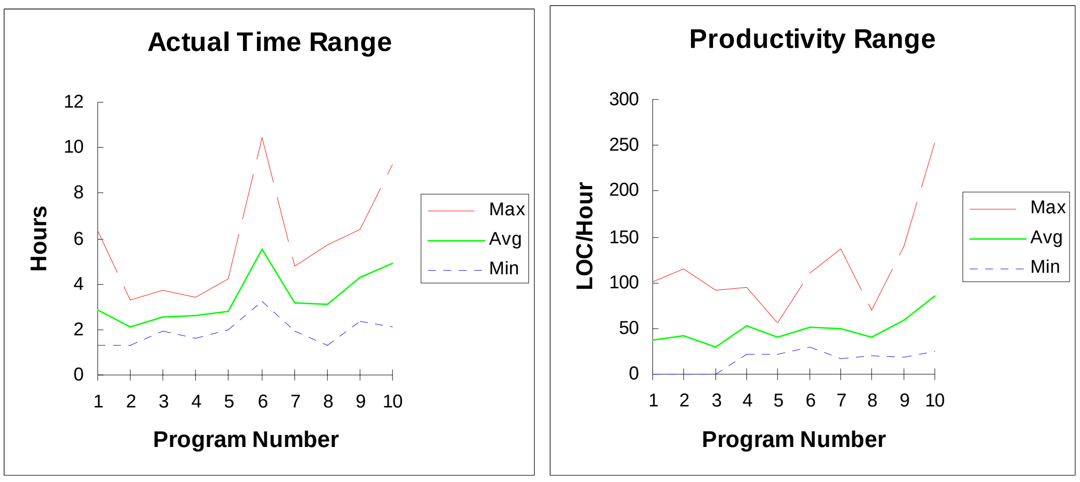
$$\text{Defect density} = \text{NumberOfDefect/KLOC}. \quad \textbf{(2)}$$

Instructors can use the defect density as an indicator to evaluate students' programming talents and understand if students have designed and implemented the assignments carefully. The students with better programming ability usually produce fewer defects, and hence, have a lower defect density. In contrast, the students with poor programming ability have a tendency to produce more defects, which can result in a higher defect density. Moreover, by considering both the defect density and productivity, instructors not only can understand the programming talents of students, but also can evaluate the maturity of students' processes. A mature process usually implies that the students have received adequate programming disciplines. Typically, the productivity and defect density of students could fluctuate from time to time in the early stage of the programming course. If the students' programming skills are improved, the software development process of the students will gradually become stable and more predictable. Thus, if the variations of both productivity and defect density become small, this may indicate that the students make progress steadily in developing their programming skills. If the productivity and defect density still fluctuate, instructors can analyze the possible causes of the variation in order to help students improve their programming skills. Further, if both the productivity and defect density of students are high, this may suggest that the students rush into implementation without an adequate design. However, if the productivity is low and the defect density is high, it could indicate that the students have serious difficulty in the program development and they require additional programming assistance from instructors.

4.2 Analysis of the Process Data for Entire Class

In addition to each student's process data, the process information for the entire class can also be insightful for instructors and students themselves. From the statistics of the process data, such as average, deviation, and regression of the time, productivity, and defect records for the entire class, instructors can assess the overall learning progress of the class. Students also can understand their own process statistics and know the differences of their programming abilities from the class average. This may point out the directions for students to enhance their programming skills or increase the student motivations.

Figure 2 shows an example of the time effort and productivity statistics for the entire class. From the average time that students spent on the assignments, instructors can understand if the efforts of students or the difficulties of assignments are proper. For example, in figure 2(a), the average time for developing programs 6 and 10 is larger than that of other programs. This may imply that the concepts for designing programs 6 and 10 are harder than that of other programs. In addition, the productivity of the entire class can facilitate the understanding of the overall performance of the class or can be used as a baseline for students to improve their own programming skills. For instance, figure 2(b) shows that the productivity of the entire class fluctuates, but the trend of the overall productivity is gradually increasing.



(a) **(b)**

Fig. 2. An example of the time effort and productivity statistics for the entire class

Moreover, from the defect data of the entire class, instructors can know the frequent defects of the class and identify students' common programming problems. Further, by sorting the defect data of the entire class according to defect fix time, those complex defects that have large fix time in the entire class can be identified. Such information can help instructors understand which programming constructs or concepts are misunderstood by most students in the class and what are the potentially serious programming problems for students in the class.

Notice that the performance or programming problems of students could be biased or misunderstood if instructors simply consider only one of the proposed process statistics. In order to obtain a better understanding of students' programming talents, instructors should attempt to consider as many process statistics as possible and take into account their teaching experiences and observations. In such a case, the process data can be very useful to instructors for designing a more effective strategy to improve their teaching quality.

5 The Architecture Design and Implementation of PDAT

In order to reduce students' overheads and increase the data quality, a supporting tool, called PDAT, is proposed to help students gather the process data of programming practices automatically. The PDAT is based on the client-server architecture as shown in Figure 3. In particular, the client of the PDAT is developed as an Eclipse plug-in so that the process data can be automatically collected during the development of assignment programs. The collected data are stored in XML format and are submitted to the server of the PDAT when students complete the programs. The server will maintain the historical process data for each student and the entire class and provide the statistical reports, as described in section 4, to facilitate the analysis of student performances. The server also maintains the assignment programs submitted by students. Thus, instructors can examine the programs, associated process data, and corresponding process statistics for each student in order to analyze the learning progresses and the programming problems of students.

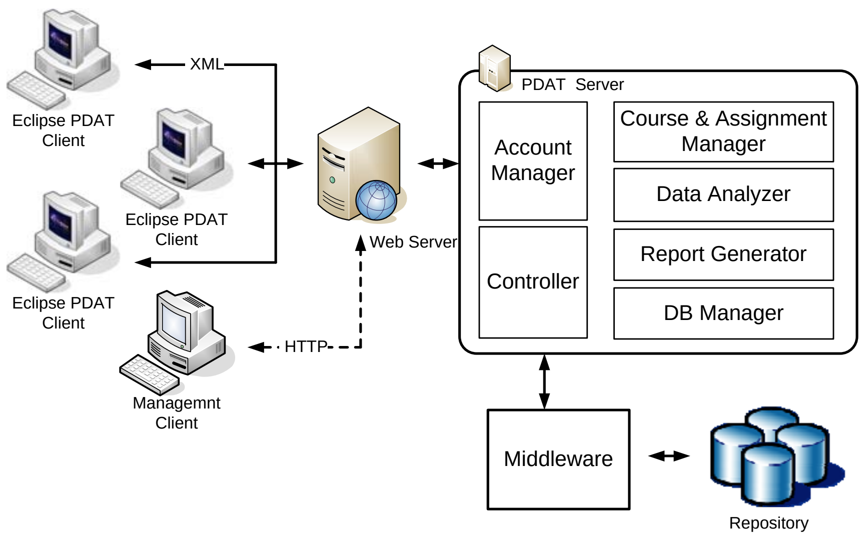


Fig. 3. The system architecture of the PDAT

Figure 4 shows the architecture design of the PDAT client and the relationships between the client and the Eclipse platform. Basically, the PDAT client consists of four subsystems. Each subsystem is described briefly as follows:

- The Data Collection Subsystem (DCS) mainly focuses on the gathering of process data that include the sizes of programs, the time efforts spent on the programs, and various defect data.
- The Controller & Plug-in Subsystem (CPS) manages the interactions among the subsystems of the PDAT client and the Eclipse platform. In particular, the CPS controls the changes of the process phases, observes various Eclipse events through the workbench and JUnit, and accesses the program information through the workspace.
- The Data Analysis Subsystem (DAS) provides the summary and reports for the collected process data, such as time statistics, productivity, and defect information.
- The Data Management Subsystem (DMS) mainly supports the management of the process data. It provides the repository for maintaining the historical process data and the functionality to upload the data to the PDAT server.

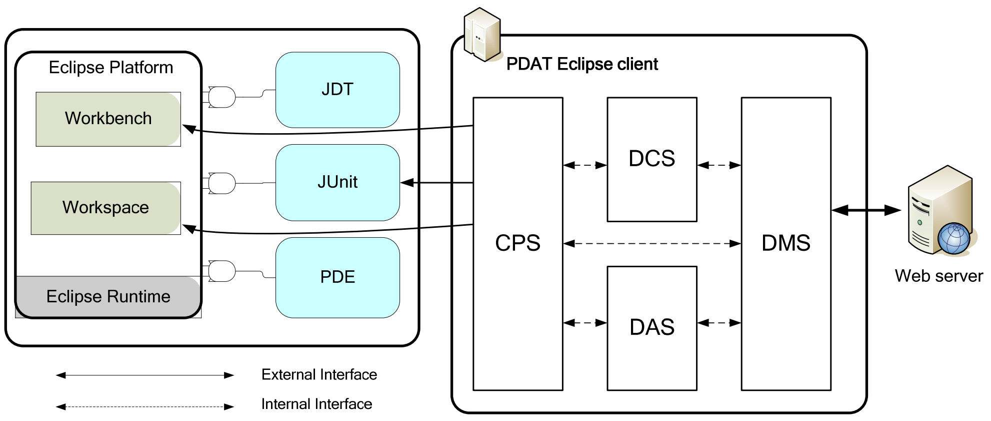


Fig. 4. The architecture design of the PDAT client

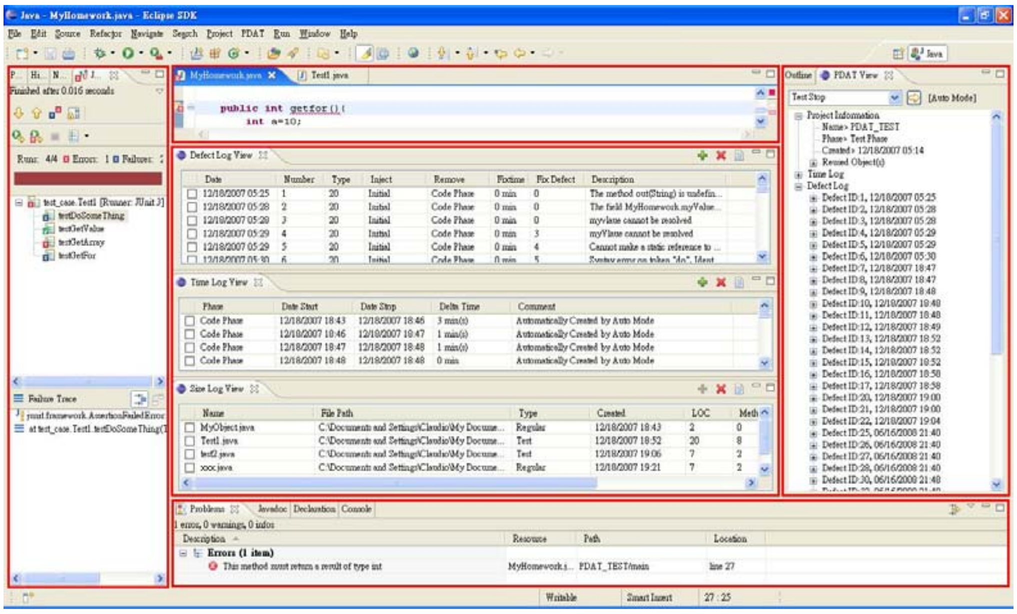
To use the PDAT client, students first need to specify the repository for storing the process data. Once the data repository is specified, the process data will be gathered automatically when the students start developing programs with Eclipse. The gathered data will be shown in the corresponding views of the PDAT. As shown in Figure 5, the collected data are presented in the size, time, and defect log views, respectively. Specifically, the size log view shows various information of each object class in the programs, such as name, location, creation time, LOC, and the number of methods. The time log view depicts the start time, stop time, and delta time (i.e., time duration) for each phase. The defect log view records various defect data of compilation and unit testing errors, such as detection time, defect type, defect injection and removal phases, fix time, and defect descriptions. Moreover, the summary of the process data is also provided and shown in the PDAT view.

After students complete their programming assignments, they can submit the collected process data to the PDAT sever through the PDAT client. The collected data are then consolidated in the PDAT server and several statistical reports about the students' processes will be generated. Figures 6 and 7 show the examples of the process statistics for an individual student and the entire class, respectively. Instructors (or students) can login the PDAT server and check the results of process analysis to evaluate the student performances.

6 Conclusions and Future Work

This paper has presented an approach to support teaching of programming courses based on the concepts of the PSP. The approach basically gathers the information about the development process of students' programming assignments, such as the sizes of programs, the time spent on the assignments, and the defect data. By analyzing the gathered process data, instructors can obtain valuable insights into students' learning progresses and understand their frequent programming problems. In addition, we have proposed several process statistics, including (1) time statistics; (2) familiarity of syntax; (3) defects hard to fix; (4) frequent defects; (5) productivity; and (6) defect density. The computations of these process statistics and their possible indications to student performances are described. Based on the analysis of the process statistics, instructors can develop a more effective strategy to improve their teaching quality.

Moreover, to reduce the overheads of data collection and improve the quality of process data, an automatic tool, called PDAT, is developed. The PDAT is based on the client-server architecture. The PDAT client is an Eclipse plug-in. It can gather the process data automatically when students develop assignment programs with Eclipse. The recorded data are submitted to the PDAT server where the data are consolidated and several statistical reports for individual students and entire class are provided. Such reports can facilitate instructors to help students improve their programming ability. In the future, we plan to apply the proposed approach to introductory programming courses, and report the experiences obtained from the experiments. Based on the experiences, we also plan to enhance the supporting tool to gather more process data, such as the time and size estimations, the size of reused programs, and the information of debugging process, and provide more process statistics in order to support the teaching of programming courses as well as the improvement of students' programming skills.



Java - MyHomework.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project PDAT Run Window Help
MyHomework.java
Test1.java
Java

```
public int getfor(){
    int a=10;
```

Finished after 0.016 seconds

Runs: 4/4 Errors: 1 Failures:

test_case.Test1 [Runner: JUnit 3]
testDoSomeThing
testGetValue
testGetArray
testGetFor

Failure Trace
junit.framework.AssertionFailedError
at test_case.Test1.testDoSomeThing(T

Defect Log View

	Date	Number	Type	Inject	Remove	Fixtime	Fix Defect	Description
☐	12/18/2007 05:25	1	20	Initial	Code Phase	0 min	0	The method out(String) is undefin...
☐	12/18/2007 05:28	2	20	Initial	Code Phase	0 min	0	The field MyHomework.myValue...
☐	12/18/2007 05:28	3	20	Initial	Code Phase	0 min	0	myvlaue cannot be resolved
☐	12/18/2007 05:29	4	20	Initial	Code Phase	0 min	3	myVlaue cannot be resolved
☐	12/18/2007 05:29	5	20	Initial	Code Phase	0 min	4	Cannot make a static reference to ...
☐	12/18/2007 05:30	6	20	Initial	Code Phase	0 min	5	Syntax error on token "do", Ident

Time Log View

	Phase	Date Start	Date Stop	Delta Time	Comment
☐	Code Phase	12/18/2007 18:43	12/18/2007 18:46	3 min(s)	Automatically Created by Auto Mode
☐	Code Phase	12/18/2007 18:46	12/18/2007 18:47	1 min(s)	Automatically Created by Auto Mode
☐	Code Phase	12/18/2007 18:47	12/18/2007 18:48	1 min(s)	Automatically Created by Auto Mode
☐	Code Phase	12/18/2007 18:48	12/18/2007 18:48	0 min	Automatically Created by Auto Mode

Size Log View

	Name	File Path	Type	Created	LOC	Meth
☐	MyObject.java	C:\Documents and Settings\Claudio\My Docume...	Regular	12/18/2007 18:43	2	0
☐	Test1.java	C:\Documents and Settings\Claudio\My Docume...	Test	12/18/2007 18:52	20	8
☐	test2.java	C:\Documents and Settings\Claudio\My Docume...	Test	12/18/2007 19:06	7	2
☐	xxx.java	C:\Documents and Settings\Claudio\My Docume...	Regular	12/18/2007 19:21	7	2

Outline | PDAT View

Test Stop [Auto Mode]

```
⊟ Project Information
    Name> PDAT_TEST
    Phase> Test Phase
    Created> 12/18/2007 05:14
  ⊞ Reused Object(s)
⊞ Time Log
⊟ Defect Log
  ⊞ Defect ID:1, 12/18/2007 05:25
  ⊞ Defect ID:2, 12/18/2007 05:28
  ⊞ Defect ID:3, 12/18/2007 05:28
  ⊞ Defect ID:4, 12/18/2007 05:29
  ⊞ Defect ID:5, 12/18/2007 05:29
  ⊞ Defect ID:6, 12/18/2007 05:30
  ⊞ Defect ID:7, 12/18/2007 18:47
  ⊞ Defect ID:8, 12/18/2007 18:47
  ⊞ Defect ID:9, 12/18/2007 18:48
  ⊞ Defect ID:10, 12/18/2007 18:48
  ⊞ Defect ID:11, 12/18/2007 18:48
  ⊞ Defect ID:12, 12/18/2007 18:49
  ⊞ Defect ID:13, 12/18/2007 18:52
  ⊞ Defect ID:14, 12/18/2007 18:52
  ⊞ Defect ID:15, 12/18/2007 18:52
  ⊞ Defect ID:16, 12/18/2007 18:58
  ⊞ Defect ID:17, 12/18/2007 18:58
  ⊞ Defect ID:20, 12/18/2007 19:00
  ⊞ Defect ID:21, 12/18/2007 19:00
  ⊞ Defect ID:22, 12/18/2007 19:04
  ⊞ Defect ID:25, 06/16/2008 21:40
  ⊞ Defect ID:26, 06/16/2008 21:40
  ⊞ Defect ID:27, 06/16/2008 21:40
  ⊞ Defect ID:28, 06/16/2008 21:40
  ⊞ Defect ID:30, 06/16/2008 21:40
```

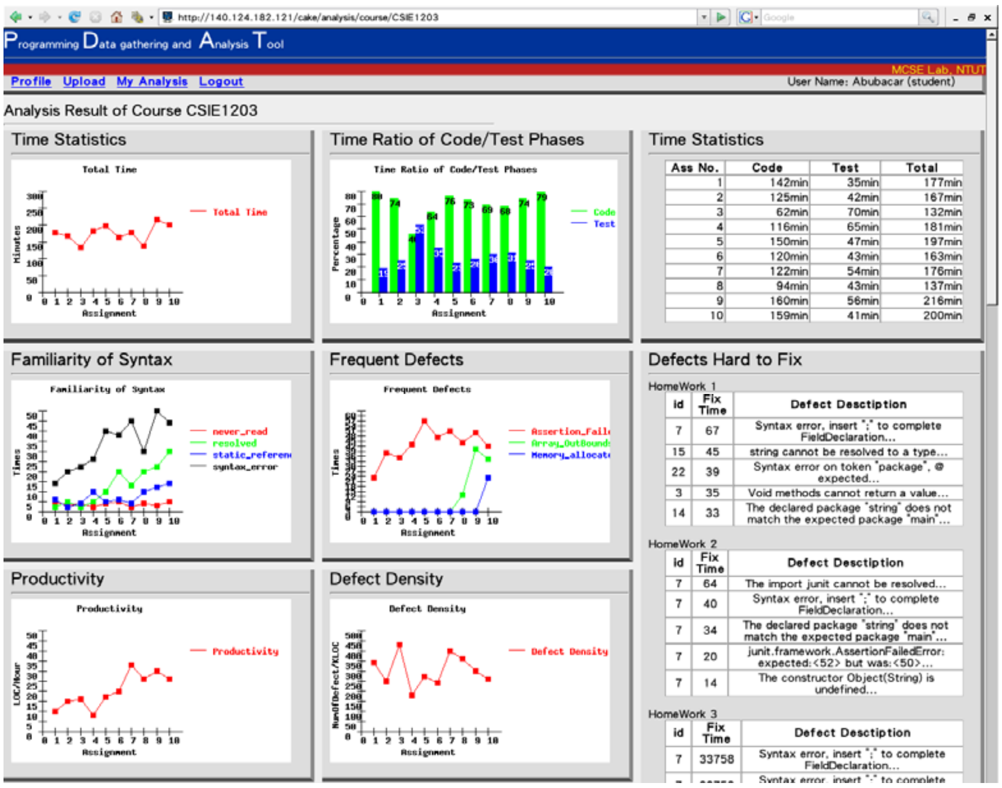
Problems | Javadoc | Declaration | Console

1 errors, 0 warnings, 0 infos

Description	Resource	Path	Location
⊟ Errors (1 item)			
This method must return a result of type int	MyHomework.j...	PDAT_TEST/main	line 27

Writable | Smart Insert | 27 : 25

Fig. 5. A screen snapshot of the PDAT client



User Name: Abubacar (student)

Analysis Result of Course CSIE1203

Time Statistics

Ass No.	Code	Test	Total
1	142min	35min	177min
2	125min	42min	167min
3	62min	70min	132min
4	116min	65min	181min
5	150min	47min	197min
6	120min	43min	163min
7	122min	54min	176min
8	94min	43min	137min
9	160min	56min	216min
10	159min	41min	200min

Defects Hard to Fix

HomeWork 1

id	Fix Time	Defect Desctiption
7	67	Syntax error, insert ";" to complete FieldDeclaration...
15	45	string cannot be resolved to a type...
22	39	Syntax error on token "package", @ expected...
3	35	Void methods cannot return a value...
14	33	The declared package "string" does not match the expected package "main"...

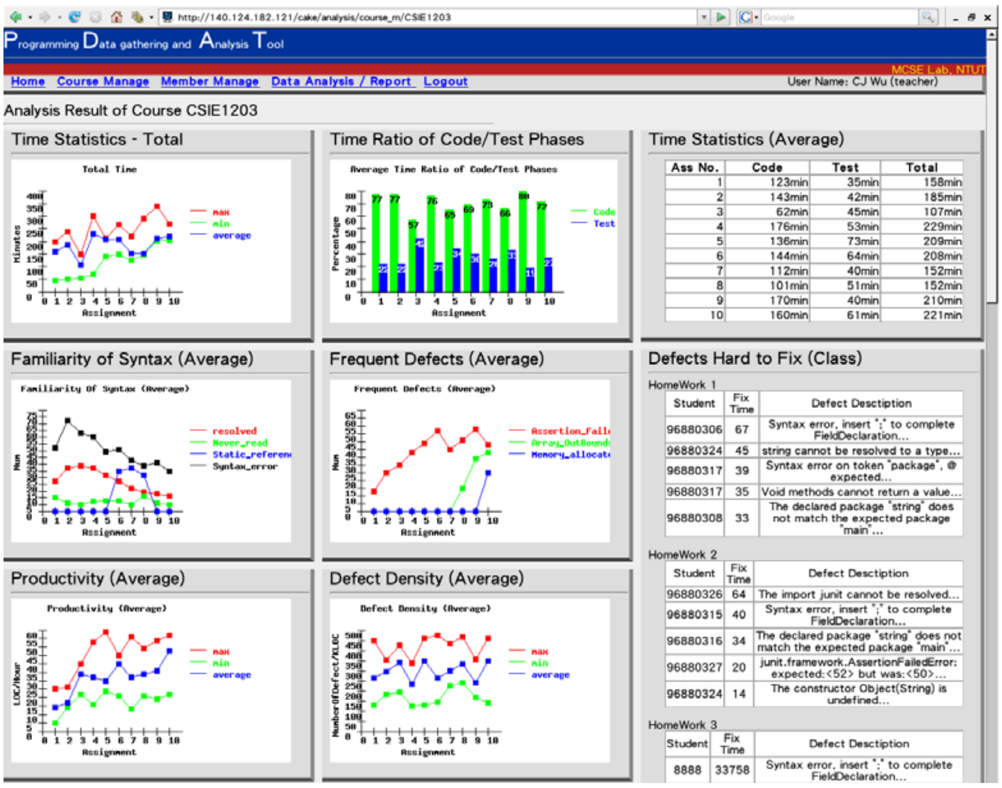
HomeWork 2

id	Fix Time	Defect Desctiption
7	64	The import junit cannot be resolved...
7	40	Syntax error, insert ";" to complete FieldDeclaration...
7	34	The declared package "string" does not match the expected package "main"...
7	20	junit.framework.AssertionFailedError: expected:<52> but was:<50>...
7	14	The constructor Object(String) is undefined...

HomeWork 3

id	Fix Time	Defect Desctiption
7	33758	Syntax error, insert ";" to complete FieldDeclaration...

Fig. 6. An example of the process statistics of an individual student


http://140.124.182.121/cake/analysis/course_m/CSIE1203
Programming Data gathering and Analysis Tool
MCSE Lab, NTUT
Home Course Manage Member Manage Data Analysis / Report Logout
Total Time
Average Time Ratio of Code/Test Phases
Familiarity Of Syntax (Average)
Frequent Defects (Average)
Productivity (Average)
Defect Density (Average)
Assignment
max
min
average
Code
Test
resolved
Never_read
Static_referen
Syntax_error
Assertion_Faile
Array_OutBound
Memory_allocat

User Name: CJ Wu (teacher)

Analysis Result of Course CSIE1203

Time Statistics (Average)

Ass No.	Code	Test	Total
1	123min	35min	158min
2	143min	42min	185min
3	62min	45min	107min
4	176min	53min	229min
5	136min	73min	209min
6	144min	64min	208min
7	112min	40min	152min
8	101min	51min	152min
9	170min	40min	210min
10	160min	61min	221min

Defects Hard to Fix (Class)

HomeWork 1

Student	Fix Time	Defect Desctiption
96880306	67	Syntax error, insert ";" to complete FieldDeclaration...
96880324	45	string cannot be resolved to a type...
96880317	39	Syntax error on token "package", @ expected...
96880317	35	Void methods cannot return a value...
96880308	33	The declared package "string" does not match the expected package "main"...

HomeWork 2

Student	Fix Time	Defect Desctiption
96880326	64	The import junit cannot be resolved...
96880315	40	Syntax error, insert ";" to complete FieldDeclaration...
96880316	34	The declared package "string" does not match the expected package "main"...
96880327	20	junit.framework.AssertionFailedError: expected:<52> but was:<50>...
96880324	14	The constructor Object(String) is undefined...

HomeWork 3

Student	Fix Time	Defect Desctiption
8888	33758	Syntax error, insert ";" to complete FieldDeclaration...

Fig. 7. An example of the process statistics of the entire class

7 Acknowledgement

This work was partially supported by the National Science Council, Taiwan, under the grant number NSC 96-2221-E-027-033.

References

[1] F. Rosales, A. Garcia, S. Rodriguez, J. L. Pedraza, R. Mendez, and M. M. Nieto, "Detection of Plagiarism in Programming Assignments," *IEEE Transactions on Education*, Vol. 51, No. 2, 2008, pp. 174-183.

[2] K. W. Bowyer and L.O. Hall, "Experience Using "MOSS" to Detect Cheating on Programming Assignments," in *Proceedings of the 29th Annual Frontiers in Education Conference*, Vol. 3, 1999, pp. 10-13.

[3] W. S. Humphrey, *A Discipline for Software Engineering*, Addison Wesley, 1995.

[4] P. Johnson and A. Disney, "The Personal Software Process: A Cautionary Case Study," *IEEE Software*, Vol. 15, No. 6, November, 1998, pp. 85-88.

[5] J. Kamatar and W. Hayes, "An Experience Report on the Personal Software Process," *IEEE Software*, Vol. 17, No. 6, 2000, pp. 85-89.

[6] Eclipse. http://www.eclipse.org

[7] S. K. Lisack, "The Personal Software Process in the Classroom: Student Reactions (an experience report)," in *Proceedings of the 13th Conference on Software Engineering Education & Training*, 2000, pp. 169-175.

[8] P. Runeson, "Experiences from Teaching PSP for Freshmen," in *Proceedings of the 14th Conference on Software Engineering Education and Training*, 2001, pp. 98-107.

[9] J. Borstler, D. Carrington, G. W. Hislop, S. Lisack, K. Olson, and L. Williams, "Teaching PSP: Challenges and Lessons Learned," *IEEE Software*, Vol. 19, No. 5,, 2002, pp. 42-48.

[10] M. Towhidnejad and A. Salimi, "Incorporating a Disciplined Software Development Process into Introductory Computer Science Programming Courses: Initial Results," in *Proceedings of the 26th Annual Conference on Frontiers in Education Conference*, Vol. 2, 1996, pp.497-500.

[11] I. Syu, A. Salimi, M. Towbidnejad, and T. Hilburn, "A Web-based System for Automating a Disciplined Personal Software Process (PSP)," in *Proceedings of the Tenth Conference on Software Engineering Education & Training*, 1997, pp. 86-96.

[12] D. Carrington, B. McEniery, and D. Johnston, "PSP in the Large Class," in *Proceedings of the 14th Conference on Software Engineering Education and Training*, 2001, pp. 81-88.

[13] J. I. Maletic, A. Howald, and A. Marcus, "Incorporating PSP into a Traditional Software Engineering Course: an Experience Report," in *Proceedings of the 14th Conference on Software Engineering Education and Training*, 2001, pp. 89-97.

[14] JUnit. http://www.junit.org/

[15] CppUnit. http://sourceforge.net/projects/cppunit

[16] B. Daum, *Professional Eclipse 3 for Java Developers*, Wrox, 2005.

[17] C.-H. Liu and Y.-C. Huang, "A PSP Eclipse Plug-in Tool for Collecting Time and Defect Data," in *Proceeding of 2006 Conference on Open Source*, Taiwan, Nov. 2006.

[18] W. S. Humphrey, "Using a Defined and Measured Personal Software Process," *IEEE Software*, May 1996, pp.77-88.