

Using the Personal Software Process to Motivate Good Programming Practices

Ralph F. Grove

Indiana University of Pennsylvania
Indiana, PA 15705 USA
rfgrove@grove.iup.edu

1. ABSTRACT

A reduced form of the Personal Software Process was used in two introductory programming courses to help students learn the value of a proper programming methodology. Students collected data during the development of their programming projects and that data was summarised and presented to the class as a whole. From the data, students were able to conclude on their own the value of early software development stages (planning, design and review) in reducing debugging time and in producing better quality software.

2. INTRODUCTION

One of the greatest challenges in helping beginning programmers to learn proper programming methodology is in convincing them that *how* one goes about developing software is just as important as *what* one produces. Due to lack of experience, beginning programmers tend towards a chaotic approach to program development which bounces among design, coding and testing in a seemingly random fashion and is often not much more than a blind search for a solution. Telling students that methodology is important and assigning reading on the topic may be informative to them but these activities do not provide them the motivation and experience necessary to appreciate the value of a formal approach to programming. A more effective approach to this lesson is to allow students to draw their own conclusions about the value of methodology based upon a quantitative analysis of their classroom experience.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ITiCSE '98 Dublin, Ireland

© 1998 ACM 1-58113-000-7/98/0008... \$5.00

The Personal Software Process (PSP) [3] is related to the Capability Maturity Model (CMM) which was developed at the Software Engineering Institute of Carnegie-Mellon University. The CMM has been used successfully to improve software development processes in a variety of organisations [1] and is now required for many U.S. defence contractors. The PSP applies the philosophy of the CMM at an individual level. It includes a well-defined process for approaching all phases of programming, a set of design techniques for programmers and a basis for data collection to support continuous process improvement at an individual level.

While the entire PSP is much too complex for beginning programmers to master, an appropriate PSP subset can be very effective in guiding students through the programming process and in motivating good programming practices. This paper describes an adaptation of the PSP to introductory programming courses and how it was used to motivate learning. Section 3 discusses the goals of using PSP and its application in the undergraduate curriculum, Section 4 presents results from actual classroom experience and Section 5 discusses conclusions, related work and the future of this research.

3. PSP IN INTRODUCTORY COURSES

Three pedagogical goals were supported by the use of the PSP in introductory Computer Science courses. The first goal was to introduce students to a proper programming methodology. The PSP stresses planning, design, design review and coding review steps in addition to the normal coding and testing steps of programming. It reinforces the performance of these additional activities for students by requiring them to document their progress through the complete process using a time log. The second goal of using the PSP was to use the time log data recorded by students as a basis for quantitative analysis of the programming process and to incorporate the results of that analysis into the curriculum in order to provide stronger motivation for students to use proper methodology. The third goal was to introduce students to the notion of continuous quality improvement, i.e., how analysis of a process can be used to improve the process over time. The PSP is designed for use in this way and provides a good basis for examining the programming process.

The complete PSP includes a significant amount of data collection, documentation and quantitative analysis, as well as the use of formal design methods. This material could fill a semester-long course of its own and is obviously too complex for beginning programmers to master. Instead of the full PSP, students were introduced to PSP basics, including time logging and code size analysis, at a level appropriate for second-year students. The PSP forms, which normally comprise several pages, were reduced to one form (Figure 1) containing the minimum data necessary to meet the goals stated above.

The reduced PSP forms include on the first page a time log and a description of seven PSP phases: planning, design, coding, review, compiling, testing and wrap-up. Students use the time log to record the amount of time spent in each of these phases each time they work on a particular project. A summary table follows the time log, in which time is summed by phase and for the complete project. The form also includes a lines-of-code (LOC) summary, which students use to record the amount of code which was copied, modified, deleted and added, as well as the total LOC in the final product. Finally, students combine the time summary and LOC summary in order to compute a productivity measure, LOC/hour.

4. RESULTS OF USING PSP IN COURSES

The PSP was introduced into two data structures courses (commonly called CS/2) using C++ and one applications programming course using COBOL. Each course included about 20 students, who were assigned 5 projects each during the course of the semester. The PSP forms were distributed with each project and were collected as each project was completed. After being entered into a spreadsheet for processing, the data were filtered to exclude submissions that included impossible data and those that were obviously completed after the fact. The following two graphs represent data collected in the applied programming class as part of programming project number four.

Figure 2 shows a scatter-plot of the ratio of minutes spent in the compile and test phases to minutes spent in the

planning, design and review phases. The graph gives a good indication of the value of planning, design and review in reducing total compile and testing time. When presented with the data in this form, students were able to reach this conclusion independently. The graph clearly indicates that students who follow the methodology and invest a proper amount of time in up-front stages of program development reap the benefit of reduced compiling and testing due to the early prevention and removal of errors. This difference has obvious implications for software quality as well.

Figure 3 shows project grades (on a scale of 0-50) vs. the ratio described in Figure 2. Points to the left end of the X-axis represent individuals who spent relatively little time in up-front activities (planning, design, and review), while those to the right end of the axis represent individuals who spent relatively more time in up-front activities. Grades were assigned before any of the PSP data was processed, to eliminate the possibility of instructor bias in this regard. Students were able to conclude from the graph that those who received poorer grades invested relatively less time in planning, design and review, a clear indication of the value of good methodology.

The above charts were presented to the class in question soon after they had completed this project, and students were asked to draw their own conclusions from the data. The group had no problems in seeing the relationships in each case and in drawing the conclusions cited above, completely on their own. Students were impressed by the significance of the data and the implied value of the PSP methodology.

The first significant problem encountered in applying the PSP at the undergraduate level were in extracting the appropriate elements of the PSP for the class in question. Several versions of the Software Development Process Log (SDPL) were tried early in the semester and students were asked to evaluate the usefulness of each version. With the help of this feedback a useable design was developed which contained the required elements in a concise and understandable form.

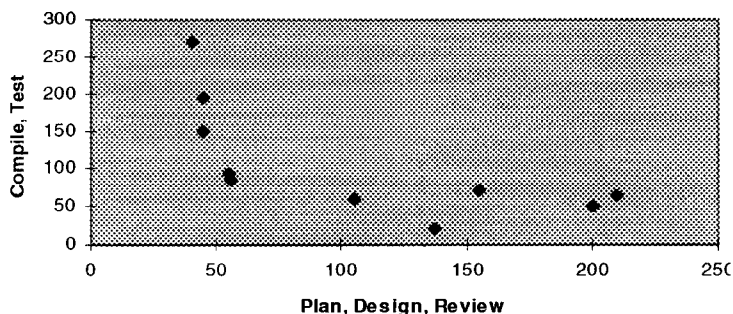


Figure 2: Compile & Test Time vs. Planning, Design & Review Time

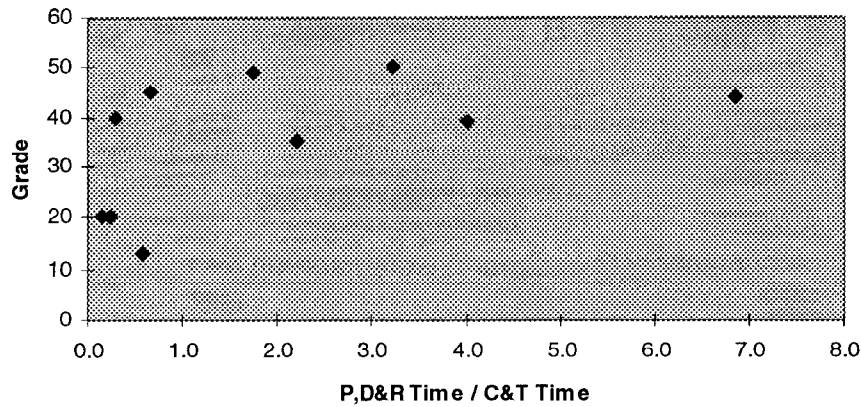


Figure 3: Grade vs. PDR/CT Ratio

Getting reliable data from the students also proved to be a problem. Several students had difficulty understanding at first the distinctions between planning, design, and review. Additional class time was spent reviewing the purpose of each process phase and a scenario-based exercise was used to clarify use of the SDPL. Some students simply rejected use of the SDPL and didn't bother to complete it or filled it in after the fact. A combination of encouragement and grade penalties was used to motivate proper use of the forms, though this continued to be a problem. Seeing the summary data collected from the forms also seemed to motivate students to use the SDPL correctly.

5. CONCLUSIONS

The most valuable outcome of the use of PSP was providing students an opportunity to discover the implications of using a proper methodology in software development by examining the data that they had collected. Specifically, the PSP summary data illustrated how proper methodology reduced program development time and improved grades. In past instances of the same classes, students have underestimated the importance of program design and review, despite testimonials from the instructor and from the textbook. Seeing indication of the value of those activities within the context of their own experience, however, provided much more credible evidence for them of the need to design and review their programs prior to coding.

Using the PSP forms also helped students to develop a better understanding early in the course of the phases of program development. The forms were a constant reminder to students of the proper steps to follow when developing a program.

Students also gained a first-hand familiarity with some fundamental software engineering principles, such as

product and process metrics and continuous quality improvement. Though these topics were not studied in depth during the courses mentioned here, it is expected that students who have used the PSP will have a stronger experience base to draw from when they do study software engineering later in the curriculum.

Hilburn and Towhidnejad of Embry-Riddle Aeronautical University have also incorporated the PSP into early CS courses, but with a different objective, that of emphasising software quality [2]. They encountered some of the same problems mentioned above, especially the difficulty of collecting data and applying the PSP method at an appropriate level.

Additional work is planned on the problem of accurate data collection, i.e., getting students to use the forms correctly and to record precise data. Possible solutions include additional exercises with the PSP forms, periodic review of the forms as they are used, and offering students bonus points for accurate data recording. Efforts will also be made to determine in a more formal way how using the PSP at this level affects student understanding and attitudes concerning the use of good programming practices.

6. REFERENCES

- [1] Herbsleb, James, David Zubrow, Dennis Goldenson, Will Hayes, Mark Paulk, Software Quality and the Capability Maturity Model, Communications of the ACM, 40/6:30-40, 1997.
- [2] Hilburn, Thomas B. and Massood Towhidnejad, Doing Quality Work: The Role of Software Process Definition in the Computer Science Curriculum. Proceedings of SIGCSE '97, pp. 277-281, 1997.
- [3] Humphrey, Watts S., A Discipline for Software Engineering, Addison-Wesley, Reading, MA, 1995.