Experience Report On The Personal Software Process In CS2

Rammohan K. Ragade, James E. Lewis
University of Louisville

Abstract

The Personal Software Process, devised by Watts Humphrey [1], is credited with increase in industrial software quality. This paper reports the results of using it on a trial basis in undergraduate education for computer engineers and computer scientists. The course chosen is the CS2 course of the ACM Computer Science curriculum on Data Structures. Students generally report improvements in their programming and software development activities during the course of four programming projects. The goal is to have a new generation of ergonomically sensitive computer and software engineers for the new millennium.

Introduction

What is PSP

The Personal Software Process (PSP) is a self-improvement process designed to help manage, control, and improve the way a person works [1,2]. The purpose of using the PSP is to improve software engineering. The PSP gives software engineers a framework for understanding their mistakes and of the importance of human factors in software development and engineering. Software errors or bugs occur increasingly in large software projects. The productivity and quality of the resulting software product has implications in the larger economical context. The PSP gives software engineers a framework for understanding their mistakes. The PSP strategy follows the approach described by Watts Humphrey:

1. Identify those large-system software methods and practices that can be used by individuals;
2. Define the subset of these methods and practices that can be applied while developing small programs;
3. Structure these methods and practices so they can be gradually introduced.

The PSP is made up of methods that include data gathering, size and resource estimating, defect management, cost of quality, and productivity analysis. The data is gathered through every phase and then analyzed once the project is completed to show error statistics that will improve the development of the next project. The data gathered will show time and estimation errors, cost performance indices, defects injected and removed per hour, process yield, appraisal and failure cost of quality, and the appraisal to failure ratio.

Several papers, workshops and conferences have shown the benefit of the PSP in industrial settings. Motorola is one of the early companies to have experimented with the PSP and found value [2]. At Embry Riddle University PSP is taught early in their Masters of Software Engineering program. They have also taught PSP in CS1 and CS2. [3]. The Software Productivity Centre (located at http://www.spc.ca/training/psp/index.htm as of October 2, 1997) teaches courses in the PSP. The Software Productivity Centre has had a 72% reduction in defects found in the testing phase, a 21% improvement in productivity, and the trained engineers can relate productivity to quality.

Need for early introduction in undergraduate education

The authors have used the PSP in the context of a large software project done by a team of 3 to 5 students over a semester. The project is a major requirement of a

senior/graduate level course on software engineering. It was observed in these projects, that many students were uncomfortable with the PSP concept. The usual argument is that they were not aware of its use in industry. Most of the students in the course have had three co-operative work assignments (coops) or are currently employed full time with an outside company.

It occurred to us that the beginning students should be introduced to PSP concepts. However, the 1 credit hour programming course did not offer the time frame necessary to inculcate PSP practices. We realized the need to explore the concept in an intermediate level. The first author teaches the CS2 course on data structures and felt that it could be tried in that course. At the time we were not aware of the Hilburn and Towhidnejad effort. We targeted the CS2 Data Structures class in Fall 1996.

The PSP should be introduced into an undergraduate's education before the undergraduate can develop bad habits. Bad programming habits are usually formed during the learning process. If these bad habits are reinforced then the bad habits become harder for the programmer to break. If undergraduates are taught early in their programming courses to use the PSP then they will form the habit of using the PSP. Then when the students are software engineers, they are more likely to continue using the PSP. Once a person has formed a habit, the habit is hard to break.

When introducing a new software process there can be many obstacles. One obstacle is to convince the professors, of the software development courses, that the process is beneficial to experienced programmers as well as novice programmers. Both authors have taught beginning programming classes. Once the professors are convinced, then the programmers need to implement the new processes. Programmers are resourceful when they try to avoid a new software process, some excuses heard are:

- "We are two busy and don't have time for processes";
- "Maybe when a new program is started but this one has been around for years";
- "Processes are busywork that no one ever reads";
- "Software development is not an assembly line, it is a creative process" [4].

Many software engineers will argue that PSP approach is impractical. There is evidence that the PSP works. By using a defined and measured personal software process, software engineers can improve their productivity and the quality of their products [5]. Other human factors to deal with is the notion of "Do I have to do it?" or "Can I cheat on some aspects of the process or can I just fake the data?". These factors are prevalent in some students. The students must be taught that the PSP is not just something to learn for a class but something to continue in all of their programming endeavors. When students see software processes being used by the software engineering community, it reinforces their learning.

Computer Science and Engineering students should see early another dimension to programming and software development activity. Victor Basili [6] and his colleagues for many years have advocated the experiential and experimental aspects of software engineering. Experiential correlation of how often programmers make errors, the types of these errors and when they begin to avoid these are important concepts. They can recognize re-usable styles of programming. Programmers should recognize factors besides the program and programming speed. They must consider the time, effort and resources it takes in the development. Errors and error rates have an impact on the applications reliability, maintainability, and dependability. A good programming practice requires documentation. Building large applications termed industrial strength applications take time and effort. Often, some of the programmers have to learn new languages, skills and concepts on the job.

A Pilot Study

The CS2 course of the ACM CS curriculum is ideal as it is well structured and

well recognized by the computer science academic community. Moreover Thomas Standish's book on Data Structures [7], has an emphasis on software principles. This book was chosen as the text book for the class so as to emphasize software principles early in the CS curriculum. The role of ergonomics as an engineering subject is increasingly appreciated. CS2 affords the possibility of making students appreciate ergonomic principles. Ultimately engineered products affect human lives. A tenet of professional ethics requires engineers to use their scientific knowledge and training to ensure the highest quality of work in products and systems. Certainly, PSP contributes to this goal by making a student aware of error patterns in software development.

Many students in the CS2 class concurrently learn C, a practice that is not officially sanctioned. Maybe they avoided the more difficult aspects of C such as memory management and pointers. In the CS2 class these are the concepts they are expected to demonstrate. So now the student has two tasks – one to learn C well and the other to learn the theoretical aspects of Data Structures. To these tasks we added a third task – to keep a personal diary to note the time and effort for doing each class programming assignment. To offset the unwillingness to participate in this effort, we decided to reserve 7% of total points for a report on how they improved their performance based on statistics from the note-book.

Yet, we do not discount the possibility that for some students, an academic assignment is a hurdle that they wish to quickly finish and go to more lucrative tasks. Some may resort to cheating – they may have falsified data. Also, their record keeping may not be consistent. While the purpose of the exercise was more of a demonstration, we did not want the process to have a major effect on the students' final grade.

Four projects were given. Programming Project 1 concentrated on all students understanding the roles of pointers and memory management. They also had projects on linked lists, stacks and queues. Project 2 was devoted to the tree data structure. Project 3 had examples from applications of graph theory. Project 4 gave them a good sample of real problems which used hashing and sorting.

Students were asked to keep a log book, which was inspected at select random times during the semester. They could earn full grade points for a careful log and a final summary report. We motivated the students by appealing to their desire to learn and improve. Not all students cared about log entries. Many had after the fact entries. These were just to show the instructor a report and earn their grade points. But there were those who took upon themselves the challenge to incorporate the methodology into their projects and learn. The comments offered by these students is encouraging to continue the effort.

From the log book the students were asked to generate comparative charts showing the time and effort they spent on *design, coding, debugging* and *testing* tasks. They could show the results as simple pie charts or histograms. They were also encouraged to submit qualitative statements indicating the types of programming and data structures specific learning that took place. Some of these comments are summarized in the Appendix.

Results and Recommendations

It is encouraging to see freshmen, sophomores and juniors making the correlation between their practices and the quality of the code they develop. They are eager to learn. As Martin Griss [8] observes recognizing design and code patterns encourages the concept of reuse – an essential concept for the new millennium. There are bound to be individuals who are not ready for the new millennium, and wish to do programming the old fashioned way of rushing to code. However, as educators it is our task to awaken their interests. It is also a wonderful way of getting freshmen, sophomores and juniors to see the correlation between ethics, professionalism and programming.

We should give more attention to training ergonomically sensitive computer engineers. When we start early in their undergraduate education, we can mold their appreciation of human factors in software quality and productivity. We believe there is room for incorporating PSP sensitivity in the first or second programming classes. Certainly, we believe there is a place for PSP in CS1 or

CS2 courses. With this training, students will not need to relearn PSP in upper level software intensive classes.

References

1. Humphrey, Watts S., A Discipline for Software Engineering Addison-Wesley Publishing Company, Reading MA, 1995.

2. Humphrey, Watts S., Introduction to the Personal Software Process, SEI Series in Software Engineering, Addison Wesley, Reading MA 1997.

3. Hilburn, Thomas B. and Towhidnejad, Massood, "Doing Quality Work: The Role of Software Process Definition in the Computer Science Curriculum". SIGCSE Bulletin. Vol. 29, No. 1, March 1997 pp. 277-281.

4. Fayad, Mohamed E., "Software Development Process: A Necessary Evil". Communications of the ACM Vol. 40 No. 9 (September 1997), 101-103.

5. Humphrey, Watts S. "Using A Defined and Measured Personal Software Process". IEEE Software. Vol. 13., No. 3, May 1996 pp. 77-88.

6. Basili, Victor R. "The Experience Factory and its Relationship to Other Quality Approaches". Advances in Computers, Vol. 41, New York NY, Academic Press 1995, pp. 65-82.

7. Standish, Thomas A., Data Structures, Algorithms & Software Principles in C, Addison Wesley Publishing Company, Reading MA 1995.

8. Griss, Martin L., "Reuse: The coming millennium offers software reuse to come into its own", Object Magazine, Sept 1997 pp. 33.

APPENDIX

Sample Observations

1. Documenting the code thoroughly and verifying certain segments, a lot of code was available for reuse. Generic segments of code could be used from assignment to assignment, such as for user -menus.

2. Using error-detecting code structures enable early correction of faulty practices.

3. Time spent on design was increased from the first two assignments, as good design rewarded with less amount of time in debugging.

4. Documenting programming practices and relating these to debugging, also encouraged spending more time on design.

5. The urge to jump to coding was toned down forcefully as the recognition of better designs before coding brought about better coding practices in the last two assignments.

6. Through the projects the programmer learned the value of modules that compile and are error free, which have few functions rather than one huge module with all the functions.

7. Pointer manipulations were better learned through careful design than through a trial and error approach in programming.

Sample notes from the diary of one student

1.

problem 3: Time used 7:00 PM -- 11:00 PM, Oct. 1
7:00 PM -- 11:00 PM, Oct. 23
7:00 PM -- 11:00 PM, Oct. 24

1. Bad argument 2 type for ListInsert(): NODE_TYPE* (NOTE_TYPE** expected)
   -- I declared 2nd argument as "**" in ListInsert(), but called the function using 2nd argument as "*".

2. Printed out wrong result
   -- I forgot to initialize Node List before using it.

3. Undefined function isdigit called
   -- Forgot including <ctype.h>

4. Infinite Loop
   --Wrong while loop condition (the condition is always satisfied).

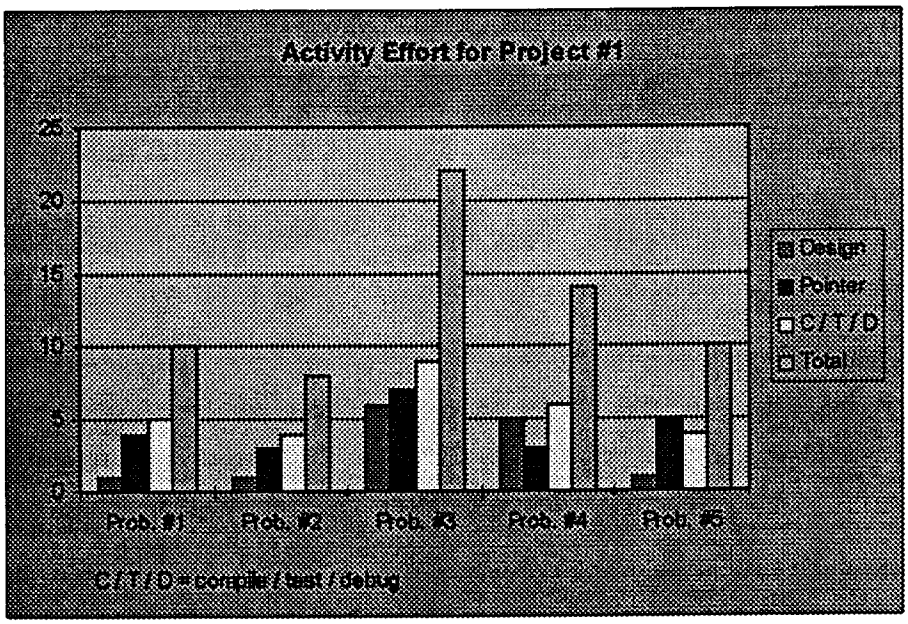
problem 4: Time used 7:00 PM -- 10:00 PM, Oct. 2

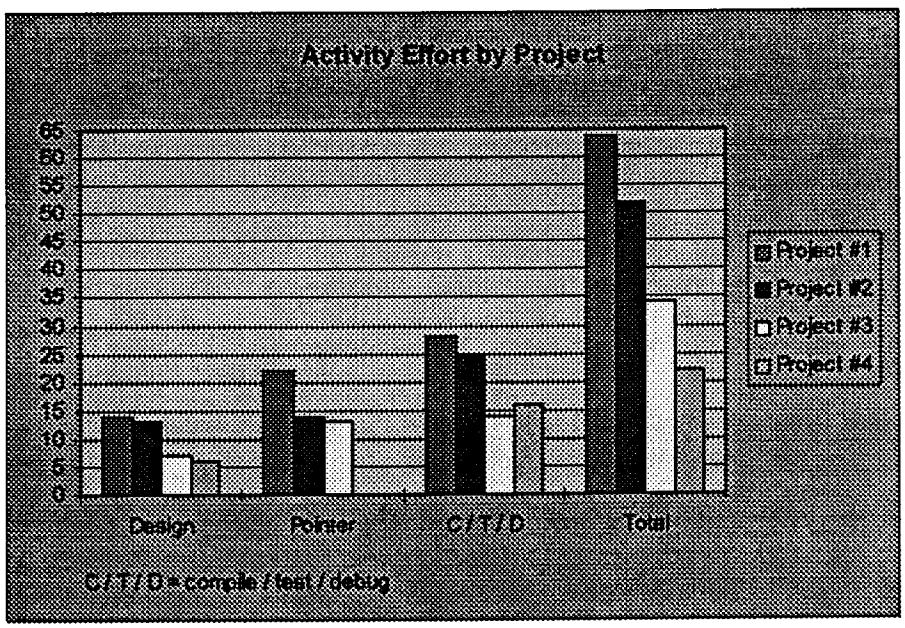
1. Segmentation fault, Bus error
   -- TreeNode T was used before memory allocation for it.
   -- Afterwards add *T = (TreeNode *)malloc(sizeof(TreeNode)) before using it, then the problem is gone.

2. Error assignment to array
   -- I used "(*T)->Airport = A" instead of "strcpy((*T)->Airport,A).

3. Function return different type
   -- In BinaryTreeSearch function, if reach successful it returns a TreeNode, but if it is unsuccessful it returns 0. Afterwards, I changed that, if reach is not successful, then it returns NULL. The program worked.

Sample Effort comparison by one student.





RAMMOHAN K. RAGADE

Rammohan K. Ragade (Ph.D. '68 Indian Institute of Technology, Kanpur, India), is Professor of Computer Science and Engineering, at the University of Louisville. He has written over 100 papers, including journal articles, refereed conference papers, chapter contributions to books. He is a co-author of a paper titled "Using simulation to aid in the design of an intelligent tutoring system", **Journal of Simulation Digest** Spring 1995 (with Pauline Cushman), for computer-aided teaching of PASCAL. His current research and teaching interests span various aspects of Software Engineering, Genetic Algorithms for Distributed Systems and Rapid Prototyping. He teaches a required course on software engineering in the MS, MENG programs in Computer Science/ Computer Engineering.

JAMES E. LEWIS

James Lewis is a 1994 graduate of Hanover College. He earned his Master of Science in Computer Science in 1996 from the University of Louisville. He is currently working on a doctorate in Computer Science and Engineering at the University of Louisville. He is employed by the Engineering Mathematics and Computer Science department as a Graduate Teaching Assistant, where he teaches programming courses in C and Pascal and math courses such as Calculus I, and Pre-Calculus. James Lewis is currently the Vice President of the student chapter of the ACM and has served as President of the chapter as well as a site coordinator of the ACM Mid-Central Regional Programming Contest. He is also participating in the pilot PFF (Preparing Future Faculty) program at the University of Louisville.