The Personal Process in Software Engineering[1]

Watts S. Humphrey

Software Engineering Institute, Carnegie Mellon University
Pittsburgh, PA 15213

[1]This work is supported by the U.S. Department of Defense

ABSTRACT
The personal software process (PSP) provides software engineers a way to improve the quality, predictability, and productivity of their work. It is designed to address the improvement needs of individual engineers and small software organizations. A graduate level PSP course has been taught at six universities and the PSP is being introduced by three industrial software organizations. The PSP provides a defined sequence of process improvement steps coupled with performance feedback at each step. This helps engineers to understand the quality of their work and to appreciate the effectiveness of the methods they use. Early experience with the PSP shows that average test defect rate improvements of ten times and average productivity improvements of 25% or more are typical.

Key Words: analysis, data, personal process, process development, process definition, process discipline, process improvement, process introduction, software engineering, software engineering education, transition.

1. OVERVIEW

The personal software process (PSP) is a technique that engineers can use to improve the predictability, quality, and productivity of their work. It has been taught to over 50 students and engineers and is being used experimentally in three industrial software organizations. Early data show average improvements of over ten times in the numbers of test defects and average productivity improvements of better than 25%. While the PSP uses various standard software engineering methods, its principal objective is to show engineers how a defined and measured process can help them to improve their personal performance. By providing immediate and explicit process feedback, engineers have the data to see what methods are most effective for them. This helps them to more consistently follow sound engineering practices.

The PSP has been developed by the Software Engineering Institute (SEI) to address the need for process improvement in small organizations and small project teams. It consists of a family of seven personal processes that progressively introduce data and analysis techniques. The engineers then use these data to determine their own performance and to measure the effectiveness of the methods they use. The basic premise is that improved personal process discipline can help to increase the effectiveness of individual engineers. [Humphrey 93, Humphrey 94] As individual performance improves, it seems likely that software team and project performance will similarly improve.

The current work on the PSP is focused on gaining experience with teaching the PSP and on identifying the most effective means for transitioning this method into industrial practice. While the PSP methods are not complex and can be readily learned through self study, early indications are that a course environment is most effective. There is also evidence that engineers are most likely to adopt the PSP methods if they work through the PSP exercises on a defined schedule and in a structured course environment.

2. THE LOGIC FOR A PERSONAL SOFTWARE PROCESS (PSP)

At root, current software development practices are nearer to a craft than an engineering discipline. The professionals have private techniques and practices which they have learned from their peers or through personal experience. Thus, few software engineers are aware of or consistently practice the best available methods. The initial PSP data show that very few engineers use such proven practices as disciplined design methods, design or code reviews, or defined testing procedures.

The introduction of improved software methods is often slow because software engineers must be personally convinced of the effectiveness of new methods before they will consistently use them. In software this is particularly true because:

- Software professionals' methods are largely private and not obvious from the products they produce. Thus, if they do not use proper methods, it is unlikely that anyone else will know.

- Software professionals are generally not trained to follow the planning and measurement disciplines needed to rigorously evaluate the methods they use.

- Evan when software groups have a common set of defined practices, these practices are not consistently followed.

- The current industrial and academic environments do not require the use of the best-known software engineering methods.

A principal issue, therefore, is how to motivate the adoption of effective methods. The PSP approach is to introduce engineers to disciplined process methods through a progressively improving personal process. The PSP provides software engineers with a defined sequence of upward-compatible personal processes that they can use to guide their individual development. Each PSP step introduces an engineering method together with a tailored exercise. By measuring their performance on these exercises, the engineers get immediate feedback on their performance. They thus see what works for them and what does not. The measurable goals for process improvement help motivate the engineers to use the methods that are most effective and to strive for further improvements. The off-line nature of the PSP exercises permits the engineers to experiment with new methods. It has been found that the PSP planning process and course schedule impose enough external pressure to provide a realistic learning environment.

The PSP approach is based on the following principles:

- By defining, measuring, and tracking their work, software professionals will better understand what they do.

- This understanding will enable the engineers to better recognize what methods work best for them and to see how they can more consistently apply them.

- The engineers will then have a defined process structure and measurable criteria for evaluating and learning from their own and others experiences.

- With this knowledge, the engineers can select those methods and practices that best suit their particular tasks and abilities.

- By using a customized set of orderly, consistently practiced, and high quality personal practices, the engineers will be more effective members of their development teams and projects.

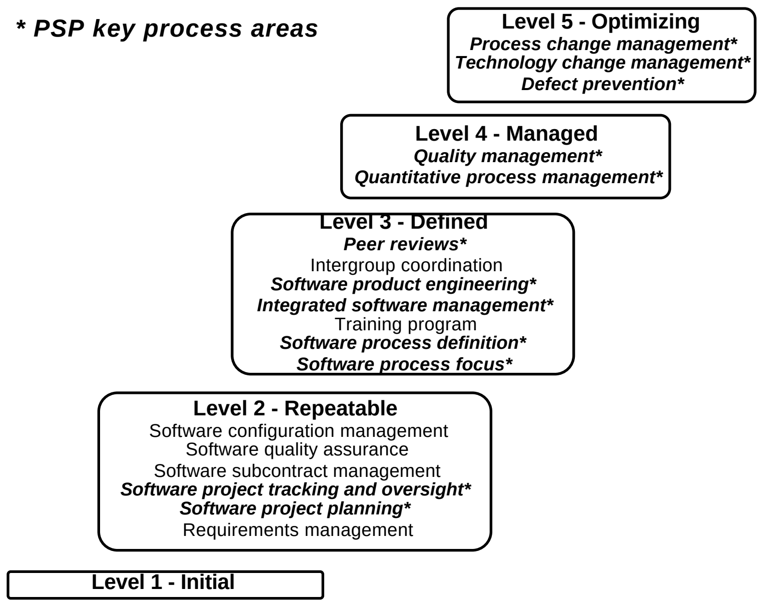
3. THE PERSONAL SOFTWARE PROCESS (PSP) STRATEGY

Today, when students start to program, they generally begin by learning a programming language. They practice on toy problems and develop the personal skills and techniques to deal with issues at this level. As they take more courses, their skills improve and they soon find they can develop fairly large programs relatively quickly. These skills, however, are inherently limited. While they may have sufficed on small-scale individual tasks, these programming-in-the-small skills do not provide an adequate foundation for the problems of large-scale multi-person projects. The PSP work follows a different strategy: it scales down industrial software practices to fit the needs of small-scale program development. By learning and using these practices on small programs, the students develop a foundation on which to build industrial-scale personal disciplines.

The Capability Maturity Model (CMM) provides a body of software engineering practices that have been found effective for large-scale software development. [Humphrey 89, Paulk 1993] Starting with the CMM, the author selected and defined a subset of the CMM key process areas for use by individual software practitioners. [Humphrey 93, Humphrey 94]

The CMM key process areas are shown in Figure 1 and those that are at least partially addressed by the PSP are shown in bold italics and noted with an asterisk. Some items are not included because their effectiveness cannot be demonstrated at the individual level. Examples are subcontract management and intergroup coordination.
Others can be usefully practiced by individuals but their implications are better demonstrated in a small team environment. Requirements management and configuration management both fall in this category. While these are both critical topics, they are more effectively introduced after the initial PSP steps have been completed. Two other key process areas more directly relate to broader organizational issues. These are software quality assurance and training programs. While the PSP capabilities are directly relevant to these areas, it is not clear what useful exercises could be developed to demonstrate them at an individual level.

Figure 1 The CMM and the PSP


* PSP key process areas
Level 5 - Optimizing
Process change management*
Technology change management*
Defect prevention*
Level 4 - Managed
Quality management*
Quantitative process management*
Level 3 - Defined
Peer reviews*
Intergroup coordination
Software product engineering*
Integrated software management*
Training program
Software process definition*
Software process focus*
Level 2 - Repeatable
Software configuration management
Software quality assurance
Software subcontract management
Software project tracking and oversight*
Software project planning*
Requirements management
Level 1 - Initial

These CMM methods have been structured in an evolving sequence of seven upward-compatible personal processes, as shown in Figure 2. Each process step is defined and used to guide engineers and students through a progressive series of process enhancements. The professionals start by using the pre-defined PSP0 process to measure their current practices. This initial process includes basic measures of the time they spend in each PSP phase and the defects they find. They then work through the seven PSP versions, adding a new software engineering method with each step. These processes are used to complete a total of ten software development exercises and to gather and analyze data on each project. Based on these analyses, the engineers improve their practices for the next exercise. This provides them explicit feedback on the effectiveness of each process step.

At the course conclusion, the engineers have learned how to measure their work, they have observed the effectiveness of various software engineering methods, and they have defined and used a special process for their own use. With this foundation, they are prepared to apply the PSP principles to other aspects of their software work.

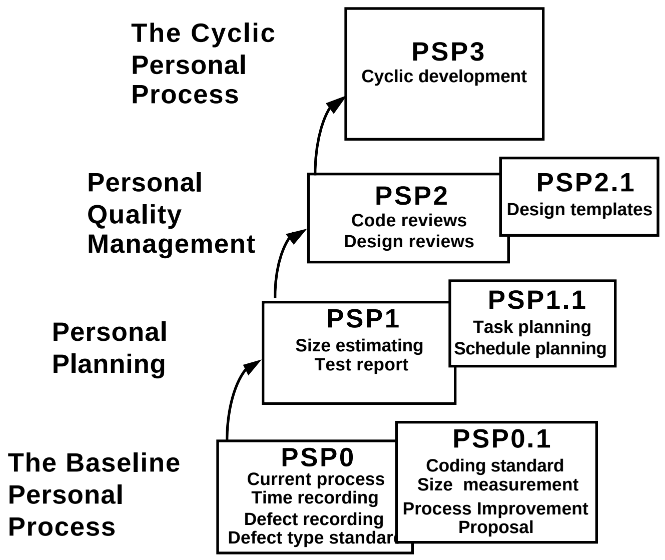
4. THE PERSONAL SOFTWARE PROCESS (PSP) EVOLUTION

Following the progression shown in Figure 2, the PSP process versions are described in the following paragraphs.

4.1 PSP0 - The Baseline Personal Process

The initial step in the PSP is to establish a baseline that includes measurements and a reporting format. This provides a consistent basis for measuring progress and a defined foundation on which to improve. PSP0 is essentially the current process the engineers use to write software, enhanced to provide measurements.

Figure 2 The PSP Evolution



Following the first programming exercises, PSP0 is enhanced to PSP0.1 by adding a coding standard, size measurement, and the process improvement proposal (PIP). The PIP provides a structured way to record process problems, experiences, and improvement suggestions. PSP0.1 also enhances program size measurement to separately count methods and procedures.

4.2 PSP1 - The Personal Planning Process

PSP1 adds planning steps to PSP0. The initial step adds size and resource estimation and a test report. In PSP1.1, schedule planning and status tracking are also introduced. The engineers are taught to:

- understand the relationship between the sizes of the programs they develop and the times they take to develop them,

- learn how to make commitments that they can meet,

- prepare an orderly plan for doing their work, and

- establish a basis for tracking their work.

While the importance of these techniques for large projects is well understood, few engineers apply them to their personal work. The PSP demonstrates the value of these methods at the personal level.

4.3 PSP2 - Personal Quality Management

An early PSP objective is to help engineers to deal realistically and objectively with the defects they inject. Programmers are often embarrassed about their errors.[2] Beizer calls this "bug guilt" [Beizer 1983]. The fact that most software errors are simple typos, oversights, or dumb mistakes generally makes engineers feel they can improve merely by trying harder. The problem is that trying harder often makes things worse. Just as in touch typing, sports, and the performing arts, the issue is inherent skills and abilities. In PSP2, a broad and consistent focus is placed on improving the engineer's ability to produce quality programs. The objective is to make quality work more natural and more consistent.

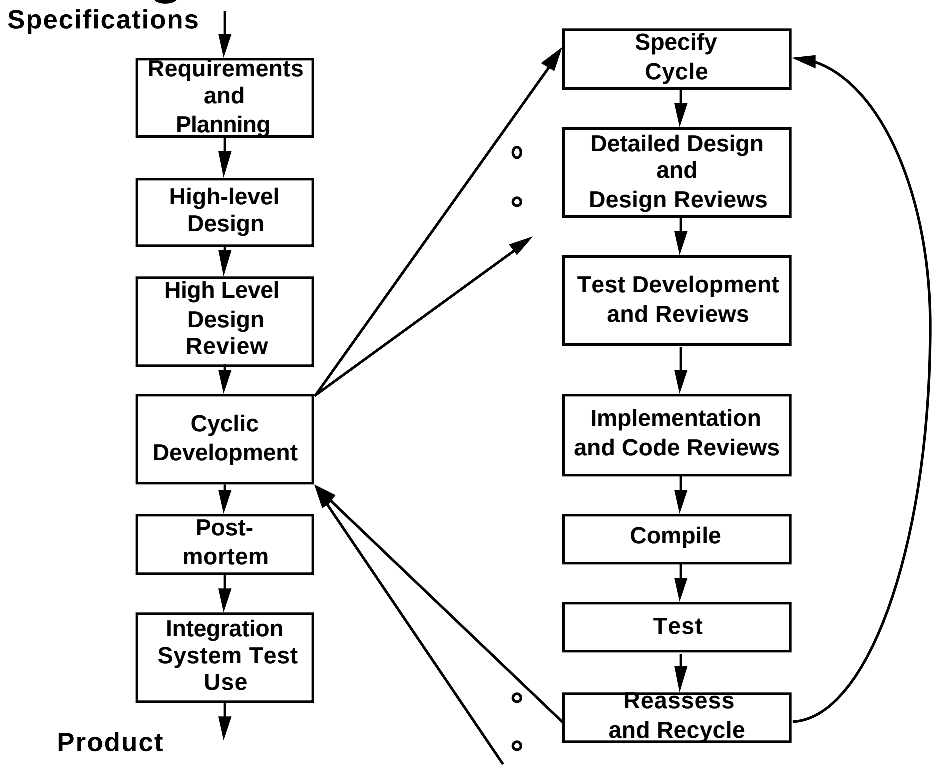
Significant improvements in engineers' defect rates are not likely unless they know how many errors they

[2]A definition note: programmers make errors or mistakes that result in program defects or faults.

make and understand their causes and consequences. From PSP defect data on both students and experienced engineers, the typical numbers of compile and unit test defects fall between 50 to 250 per thousand lines of code (KLOC). This is for uninspected and unreviewed programs.

documentation development, and test development. Phase entry and exit criteria are important because without them, it is difficult to do effective phase reviews or to crisply define development status.

Figure 3 The PSP3 Process


Specifications
Requirements and Planning
High-level Design
High Level Design Review
Cyclic Development
Post-mortem
Integration System Test Use
Product
Specify Cycle
Detailed Design and Design Reviews
Test Development and Reviews
Implementation and Code Reviews
Compile
Test
Reassess and Recycle

PSP2 adds personal design and code reviews to PSP1. These reviews help the engineers to find defects earlier in their processes and to see the benefits of finding them early. They analyze the defects they find in the early programs and use these data to establish review checklists that are tailored to their personal defect experience.

The design process is addressed in PSP2.1. The intent is not to tell engineers how to do design but to address the criteria for design completion. That is, when they have finished the design, what must they have? In PSP2.1, design completeness criteria are established and various design proving and consistency techniques are examined. While the design phase is used as an example of completeness criteria, the same approach can be used with the other process phases. Examples are requirements specification,

4.4 PSP3 - A Cyclic Personal Process

To this point, the PSP has concentrated on a linear process for building small programs. The PSP3 process introduces methods for individuals to use when they are developing larger-scale programs. It is still focused on the individual, however, and does not deal with the communication and coordination problems that are an important part of larger-scale system development.

In scaling the PSP2 up to larger projects, the approach is to subdivide the personal process of developing larger programs into PSP2-sized pieces. These larger programs are then designed to be developed in incremental steps. The way this is done by PSP3, the cyclic development process, as shown in Figure 3. The first build is a base module or kernel that is enhanced in iterative cycles. In each iteration, a complete PSP2 is used, including design, code, compile, and test. Since each enhancement builds on the previously completed increments, the PSP3 process is suitable for programs of up to several KLOC.

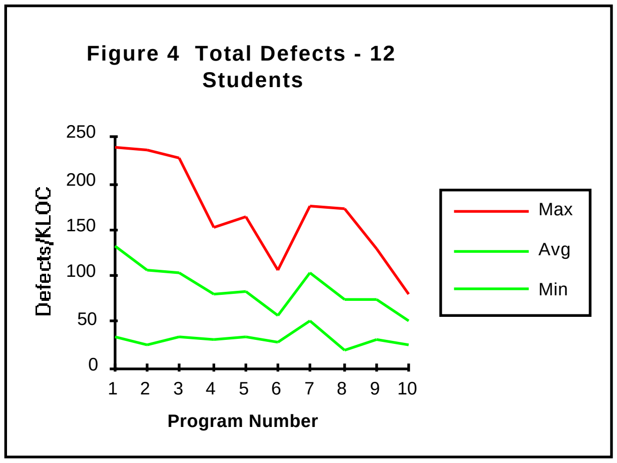


Figure 4 Total Defects - 12 Students

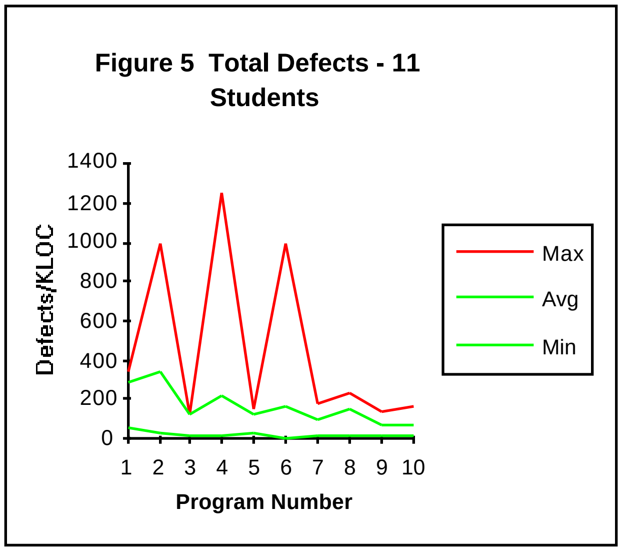


Figure 5 Total Defects - 11 Students

The cyclic PSP3 process can be an effective element of a large-scale development process only if each successive software increment is of high quality. The engineers can then concentrate on verifying the quality for the latest increment without being troubled by defects in the earlier cycles. If a prior increment has many defects, however, testing will be more complex and the scale-up benefits will be largely lost. This is one reason for emphasizing design and code reviews in the earlier PSP steps. The engineers can also use the test report to facilitate rerunning earlier tests to check for regressions.

4.5 TSP - The Team Software Process

As project size increases beyond several KLOC, there is a point at which team projects are more appropriate. The PSP briefly introduces the concepts for coupling individual PSPs into larger team activities but it does not define team processes or perform team exercises.

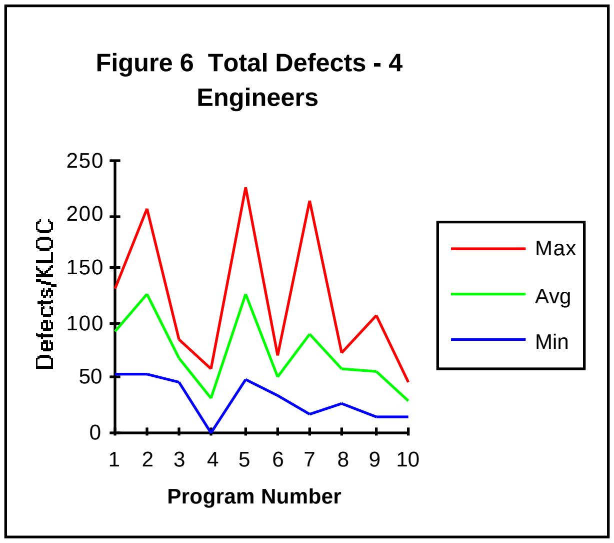


Figure 6 Total Defects - 4 Engineers

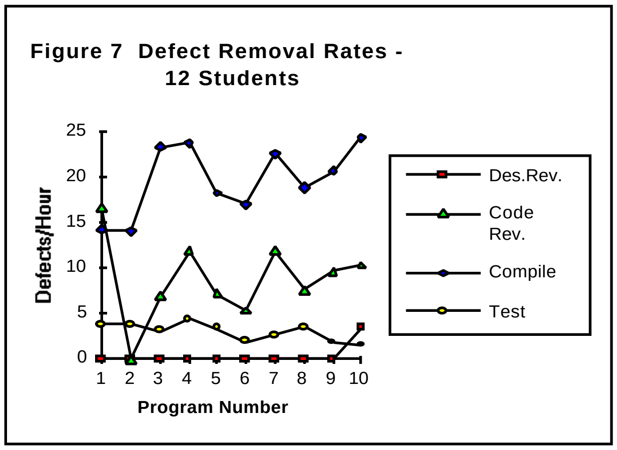


Figure 7 Defect Removal Rates - 12 Students

5. PSP EXPERIENCE

A textbook manuscript draft has been used to teach PSP methods in two university courses in the Fall of 1993 and four more courses in the Spring of 1994. [Humphrey in press] The data from these courses indicates that both students and experienced engineers can get substantial benefits from using PSP methods.

The defect data for a class of 12 graduate software engineering students at Carnegie Mellon University

(CMU) are shown in Figure 4. These students were required to have several years industrial experience before they could register for the Masters in Software Engineering program at CMU. At the beginning of the course, their defect levels initially ranged from slightly under 50 to nearly 250 defects per thousand lines of code (KLOC). These numbers are this large because they include all the defects found in desk checking, compile, and test.

Figure 8 Defects Found in Test - 12 Students

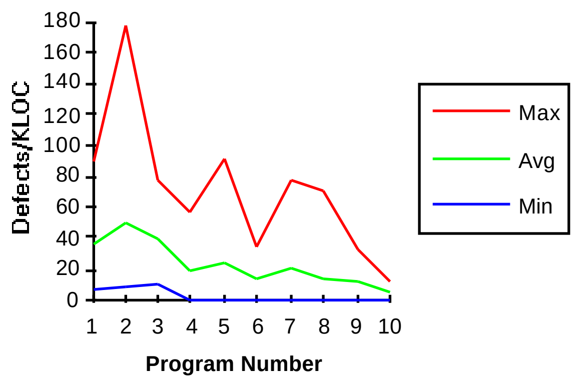


Figure 9 Defects Found in Test - 4 Engineers

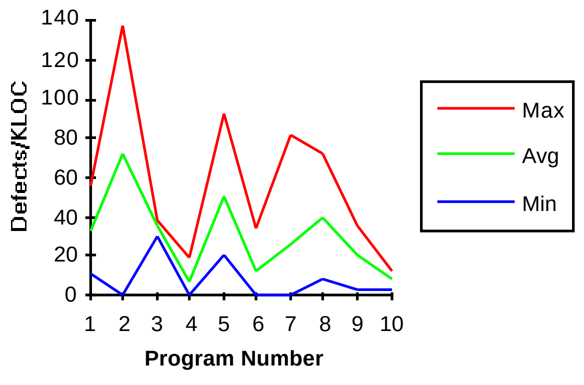
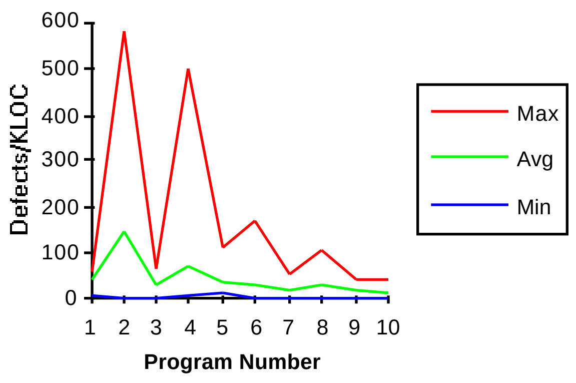


Figure 10 Defects Found in Test - 11 Students



Before they achieve language fluency, beginning programmers may have even higher defect rates. This is shown, for example, from the data for a class of 11 less experienced students in Figure 5. Their defect levels averaged 200 per KLOC at the beginning of the course with some numbers over 1000. At the end of the course, their defect numbers were also significantly reduced. In Figure 6, a group of 4 experienced engineers in a different course averaged around 100 defects per KLOC. These engineers all had over ten years of industrial experience before taking the course. Another group of 8 experienced engineers who have not yet completed the course, averaged 150 defects per KLOC at the beginning of the course with a high of 458 defects per KLOC. While their rates are substantially improved during the first half of the course, their final data are not yet available.

While many organizations report much lower defect numbers, these do not generally include the defects found in desk checking compiling, or even unit test. With the PSP, the engineers count all their defects.

One of the principal PSP measures is the defect removal rate. Figure 7 shows the average defects per hour for 12 students for design reviews (DR), code reviews (CR), compile (C), and test (T). The students are generally surprised to see how ineffective their testing time is at removing defects. They learn, for example, that in one hour, they can find an average of 10 defects in code reviews and only 2 or 3 in test.

Throughout the course, the students gradually learn improved methods to detect and prevent defects, resulting in significant over-all improvements in defect levels. Figure 8 shows the average improvement in test defects for 12 students. The average test defect levels in unit test improved from 36 per thousand lines of code to 3.9. This is a class average of about ten times improvement. Both the highest and lowest individual defect levels also improved substantially. Similar results are shown for the class of 4 experienced engineers in Figure 9 and the 11 student class in Figure 10.

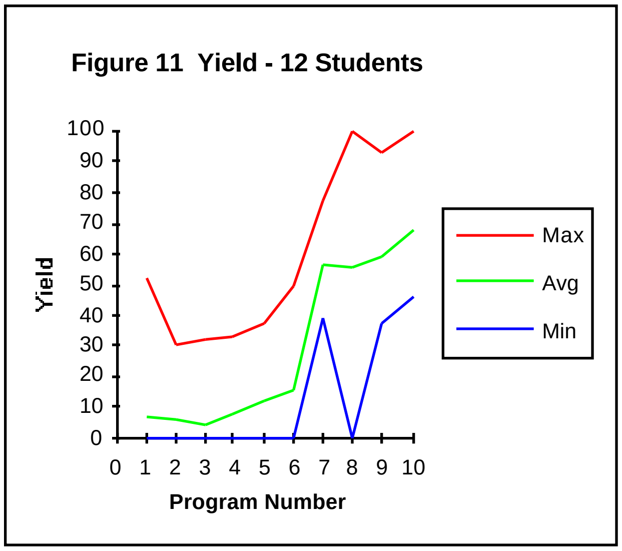
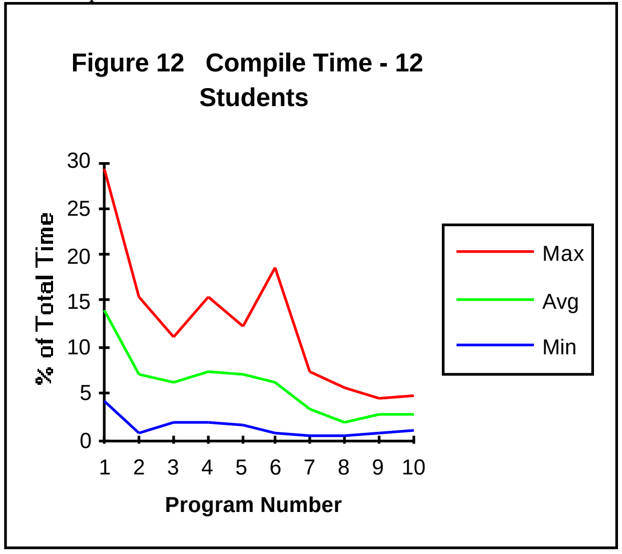


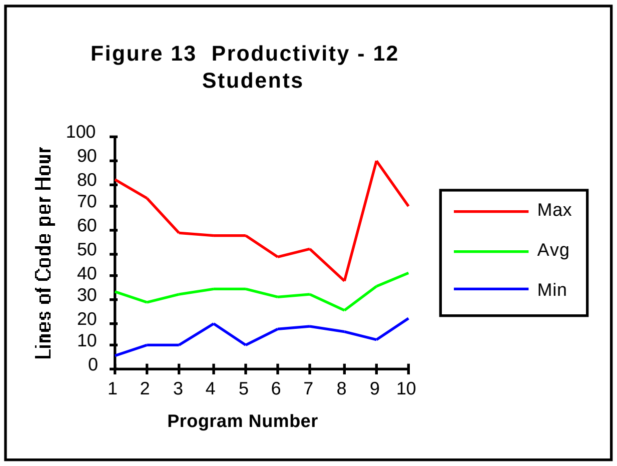
Figure 11 shows that this major improvement resulted in part from a PSP concept called yield management. One principal objective of the course is to prevent or remove defects before the first compile. Through appropriate process measurements, the students become aware of the numbers and types of defects they inject. As Figure 12 demonstrates, this resulted in a substantial reduction in compiling time. Also, in Figure 13, average productivity increased by about 30%. Note, however, that productivity improvement is not the key concern of the PSP. The principal focus is on improving the predictability and quality of the work. A key PSP objective is to do this without significantly reducing productivity.

While this is only a limited sample, it appears that the more experienced students make the greatest improvement. Indications are that the less experienced students spend more time learning how to write the programming exercises and less on process improvement. The more experienced engineers are able to focus on process improvement and learn more effectively from the data they gather.

6. PSP INTRODUCTION STRATEGIES

There are two PSP introduction strategies. One starts by introducing these methods in university curricula. This is initially being done at the graduate or senior undergraduate level. At a later time, the methods should be introduced with beginning programming courses. Such an introduction should, however, be done in concert with basic programming methods and techniques.





Because it will likely take many years to introduce these methods through the educational system, the SEI is also exploring means for directly transitioning these methods into industrial practice. One of the three industrial organizations currently involved in this work is the Digital Equipment Corporation, where several projects are participating in an introduction program. The Hewlett Packard Corporation is also using a course format to train 20 engineers in PSP methods. The AIS corporation has just completed training 8 engineers in PSP methods and they are currently applying these methods to their software work. While experience data are not yet available from any of these organizations, early evidence indicates that the engineers are finding the PSP approach useful to them in their work.

7. CONCLUSIONS

The early work on the PSP indicates that a structured, disciplined, and measured personal software process can provide the guidance and feedback needed to help engineers improve their personal performance.

- The engineers are more aware of their work and better able to understand areas where they can improve.
- The process measures provide them direct and explicit feedback on their performance.
- This rapid feedback reinforces their use of sound engineering practices.
- A course format has been found most effective for introducing the PSP in industrial organizations.
- Several engineers have attempted to use PSP methods in their regular work. Without the rapid feedback provided by the PSP exercises, however, they have been unable to sustain the needed process discipline.

The PSP is not a magic answer to the problems of developing good software. The methods take time and effort to learn and they require consistent discipline to use. While the initial PSP work concentrates on the design, code, and test phases of software development, the PSP principles can be applied to requirements specification, product maintenance, test planning, documentation development, or many other aspects of the software process. The detailed design, code, and unit test phases were selected because they are important development phases and they are most suitable for the small but challenging classroom exercises.

The PSP work does not rely on any special tools or aids. Initially, such standard aids as word processors, spread sheets, and statistical packages provide an adequate base for PSP use. An early attempt was made, however, to design a simple tool to assist in time and defect recording. It was found that the tool was actually less convenient than pencil and paper. It is thus clear that, to be most helpful, PSP support must be integrated into the working environment. If done properly, such support would make the PSP methods more efficient and easier to use. With CASE facilities to automatically log time, track defects, maintain data, and present statistical analyses, the PSP likely would be easier to learn and more efficient to use.

ACKNOWLEDGMENTS

Richard Barbour, Howie Dow, Susan Garcia, Mark Paulk, and Larry Votta have kindly reviewed this paper and I am much indebted to them for their helpful comments and suggestions.

REFERENCES

Beizer, B. (1983), *Software Testing Techniques.* Van Nostrand Reinhold, New York, NY.

Humphrey, W.S, (1989), *Managing the Software Process.* Addison-Wesley, Reading, MA.

Humphrey, W.S. (1993), "The Personal Software Process, Rationale and Status," *The 8th International Software Process Workshop*, Wadern, Germany.

Humphrey, W.S. (1994), "Process Feedback and Learning," *The 9th International Software Process Workshop*, Airlie, VA.

Humphrey, W.S, (in press), *A Discipline for Software Engineering.* Addison-Wesley, Reading, MA.

Paulk, M.C., Bill Curtis, M. B. Chrisis (1993), "Capability Maturity Model for Software, Version 1.1," Technical Report, CMU/SEI-93-TR-24, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.