Title: Learning the PSP

Results and opinions from a diverse group of experienced software engineers in Ireland

Patrick O'Beirne (*Systems Modelling Ltd)*
Joc Sanders *(Centre for Software Engineering)*

Abstract

The objective of the Personal Software Process (PSP) is to help individual software engineers to improve the quality, predictability, and productivity of their work. Humphrey's textbook [1] provides a defined sequence of process improvement steps coupled with performance feedback at each step. This helps engineers understand the quality of their work and to appreciate the effectiveness of the methods that they use. There is published evidence of improvement in academic courses. The Centre for Software Engineering (CSE) based on the campus of Dublin City University in Ireland organised a PSP course to investigate its applicability to software engineers in Irish industry. A study cluster of fourteen experienced industrial software development professionals followed a twelve-week PSP course led by Patrick O'Beirne of Systems Modelling Ltd.

The course consisted of a half day a week lecture and discussion and a half day at work carrying out assignments consisting of a programming exercise and data collection on their process metrics. The participants used their own various programming languages and environment. Ten completed the course and assignments, a 70% completion rate. This experiment in introducing PSP in Ireland shows a four times decrease in test defect rate with no impact on productivity. The participants learned that code reviews were three times more effective at reducing defects than testing. There was no obvious improvement in estimating skills with this group who already had an average of five years experience. This report quotes comments from the participants on their attitude to time recording, estimating, reviews, and defect management. Their evaluations showed that 70% now enthusiastically recommend design and code reviews to their management, and intend to build up a defect database to support reviews. Almost as many consider time recording & estimation valuable. They all stressed the need for more automated and tool support, and research into this will be ongoing.

Keywords

Personal, Software, Engineer, Process, Improvement, Quality, Productivity, Defect, Management, Training, Planning, Estimating, Scheduling, Design, Code, Reviews.

Contents:

1. The PSP

The PSP was developed by the Software Engineering Institute (SEI) based at Carnegie Mellon University in Pittsburgh, USA. It is being taught in a variety of academic and industrial environments. It aims to provide the individual discipline to underpin the organisational progress towards process improvement described by the CMM (Capability Maturity Model). PSP is described at length in Watts Humphrey's text books [1],.[4]

Table 1 shows the various stages of the PSP and the corresponding level of organisational maturity in the CMM model that they support and in turn would be supported by.

PSP Level	Process Contents	CMM Practices
0	The Baseline process plus Time Recording. Defect Recording & Defect type standard	Software project tracking & oversight
0.1	Program Size measurement Coding Standard Process Improvement Proposal	Defect Prevention Technology Change management
1	Program Size Estimation Test Report	Integrated Software Management
1.1	Resource and Schedule Estimation & Planning	Software project planning
2	Design & Code Reviews	Software Process Definition Software product engineering
2.1	Design Templates	Quantitative Process Management Quality Management Process Change Management
3	Cyclical process for larger programs.	Software Process Focus

Table 1

Current practice

The conventional image of software development is illustrated by a quote from an article by Tom Golden in the IT Post of Sunday 30 March 1997: "System development is inherently chaotic and it cannot be thoroughly planned". Most organisations are at CMM Level 1, using a fire-fighting approach to scheduling and testing. According to Watts S. Humphrey[1], "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. Their methods are not obvious from the products they produce, so if they do not follow proper methods, it is unlikely that anyone else will know. They are generally not trained to follow the planning and measurement disciplines needed to rigorously evaluate the methods they use."

The crux issue is how to motivate the adoption of effective methods. The key to the PSP is a self-convincing method that reveals to engineers the hard data about their planning, productivity and defects. They get immediate feedback on process improvements and thus see what works for them and what does not.

Definitions

Activity: A development action such as Design, Code, Compile, Review, or Test.

Phase: The PSP operates a strict waterfall model. A phase starts with the first occurrence of an activity. and stays there until the first occurrence of the next activity. For example, you start in the design phase, and it ends when you type the first line of code. Coding ends when you hit the compile key first. Testing starts when you first run the program. The rationale here is that code written in later phases (e.g. testing) is really fixing defects found in the earlier phases

LOC : Lines of Code (typically C or C++) are used as a basic size measurement. Lines of code taken unchanged from previous work are counted separately from lines inserted and edited. The latter are referred to as "New & Changed LOC" and are what are counted in productivity figures.

2. The PSP Coaching Cluster

CSE's objectives in organising the PSP Coaching Cluster were firstly to build awareness of the potential benefits of PSP among Irish software developers, and secondly to gain experience of delivering PSP training in an industrial (rather than academic) setting. It was advertised to potential participating companies as an opportunity to evaluate the significance of PSP for themselves, while training up to 2 experienced software engineers in PSP disciplines.

High drop-out rates have been reported by a number of other PSP trainers, so considerable attention was given to building commitment to complete the course, both among individual participants and their managers. Before students were asked to commit themselves to taking the course, they attended a briefing session which fully explained the objectives, what would be involved, and the amount of effort they would need to put in (underestimated at 1 full day a week). Before acceptance on the course they were required to sign a 'contract' committing themselves to complete the course, and to obtain their manager's countersignature, agreeing to provide an appropriate degree of support. This approach appears to have been successful. Of the 14 participants from 10 companies who started the course, 10 from 8 companies completed it, finishing 6 or all 7 of the course assignments, giving a completion rate of 70%.

The course itself consisted of 12 weekly sessions between November 1996 and February 1997, with a break for Christmas/New Year holidays. It followed the format of Humphrey's original course quite closely, with students attending 3 hours of lecture/discussion each week, and completing assignments in between. Table 2 shows the schedule of lectures and assignments.

Date	Unit	SUBJECT	LECTURE	ASSIGNMENT
24-Oct	**0**	Introduction	Introduction	Commitment
7-Nov	**1** PSP0	1:Overview 2:The baseline process	Definitions of Process, Phases, PSP Scripts, Standards, Forms. Project Plan, Defect Type Standard, Time and Defect Recording logs	Read PSP0 script, appendix C WSH. Write program 1 (file I/O) using PSP0 to estimate total time
14-Nov	**2** PSP0.1	3:The Planning Process 4: Measuring Software Size	LOC accounting, coding standards, Scripts, Post-Mortem, PIP	Read PSP0.1 script R1: write your LOC counting standard. Write program 2 (file edit) using PSP0.1 to estimate time by phase and New&Chg LOC.
21-Nov	**3**	5:Estimating Software Size I	review forms, scripts	R2: Write your own coding standard. Use PSP0.1 for program 3B (error check) to estimate LOC.by type and record test results
28-Nov	**4**	PSP1 5:Estimating Software Size II	The PROBE method, Explain the Size Estimating Template and Test Report template	Use PSP 1 for Program 4 (more error handling)
05 Dec	**5 PSP1**	6: PSP 1.1 Resource and Schedule Estimating	A Planning framework, Schedule estimates. Task and Schedule planning. Earned value tracking	R3: defect analysis report. Use PSP1.1 for Program 5B (files of arrays)
12 Dec	**6**	7: Measurements in the Personal Software Process	Goal-Question-Metric paradigm, the PSP Spreadsheet	R4: Midterm report and halfway course feedback report. Significant effort.
09 Jan	**7**	8: PSP2 Code Reviews	Review methods, Peer review, Explain defect removal measures	Prepare DR & CR checklists.
16 Jan	**8**	8b: Design Reviews	Design methods	Write Program 6B Regression on file data requires A7.
23 Jan	**9**	9: Software Quality Management	The Economics of defect removal, Yield management, benchmarking.	Use PSP2 for Program 7B: Linear regn. Parameters & prediction interval w/o 5A using Des Review checklists req A8
30 Jan	**10**	10: PSP2.1 Software Design	Notations and Templates: Functional, State, Logic, Object Oriented.	Implement program 7B using Code Review checklists.
06 Feb	**11**	11: Scaling up PSP 13: Process classification.	Process Frameworks. People matter.	Start writing R5, Final report, also report to CSE and your management on implementation.
13 Feb	**12**	14: Using the Software Process in an organisation	Making commitments, cost & benefits, mentoring	Feedback on course, present final reports.

Table 2

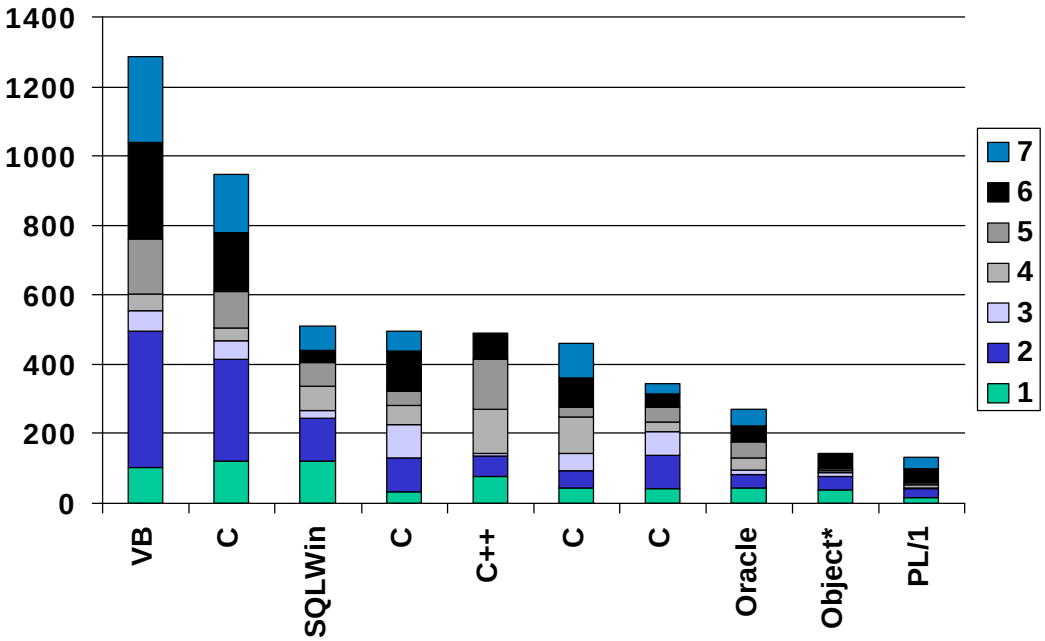
The Participants.

Participants' programming experience ranged from 2 to 10 years with an average of 4.8 years. Their languages, total Lines of Code written, and total PSP development time varied greatly, as shown at Table 3 and Figure 1. Most wrote around 500 LOC in total.

Exercises Completed	Total time (hours)	Total LOC	Average LOC per hour	Language
7	25	1288	52	Visual Basic
7	16	948	59	C
7	13	510	38	SQLWindows
7	23	495	21	C
6	13	491	37	C++
7	20	461	23	C
7	24	344	14	C
7	13	269	21	Oracle Forms3
6	16	143	9	ObjectStar
7	30	131	4	PL/1

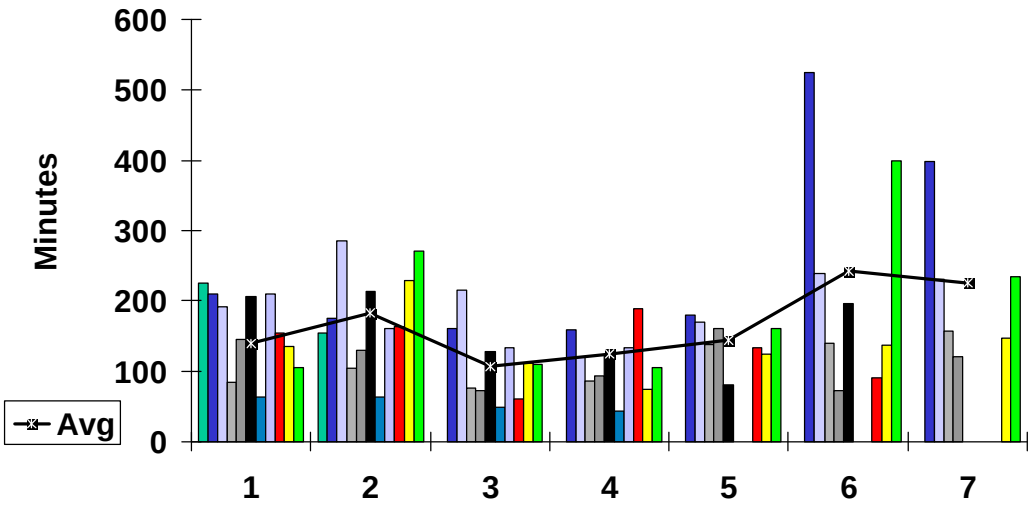
Table 3 - Total LOC for 10 participants for all 7 exercises

Fig. 1: Total LOC: 7 assignments, 10 students



Most took about 2-3 hours to do each exercise, except for a few who had trouble with the last two; see Figure 2. They also spent the same length of time studying the text. Taken with the 3 hours attendance at the cluster sessions, that is about 8 hours a week, or a 100 hour course in total. The full PSP is expected to take 150 hours, so this was a compressed version.

Fig. 2: Time for each assignment



The author has not seen what any application looks like - the PL/1 app runs on a mainframe. Unlike college student courses, the PSP course presenter was not an assessor of the final product - that was left to their professional judgement. The focus as course leader was on the process as they documented it and reported back via the Project Report sheets.

Section 3 presents the results achieved by the participants during the course.

A review of the progress of the cluster was held with the participants at the mid-point. As a result some useful improvements to the style of delivery were made, in particular increasing the amount of group discussion, while reducing lecture time, and getting students to read ahead in Humphrey's text book.

At the end of the cluster, participants were asked to complete a questionnaire designed to elicit their opinions on PSP, and their responses were discussed in a final review of the cluster. Section 4 presents an analysis of the opinions expressed by the participants.

3. Participants' PSP Data

Time & Size estimating:

Figure 3 is a chart of all the time estimating errors showing that most are within +- 50% ; the standard deviation is +/- 36%. There is a standard deviation of +-70% in New & Changed LOC Size estimation, see Figure 4. So they are more accurate - probably because more experienced - at time than size estimation. They could also be "time-boxing" - that is, doing what they know can be fitted into the typical two-hour assignment time. The loose exercise specifications allow design flexibility which gives this kind of choice. So for example, in calculating a "Student t" value, they could either build up a table and do a lookup with the parameters "number of variables" and "confidence"; or they could simply ask the user to input it.

Fig. 3: Time Estimating Error%

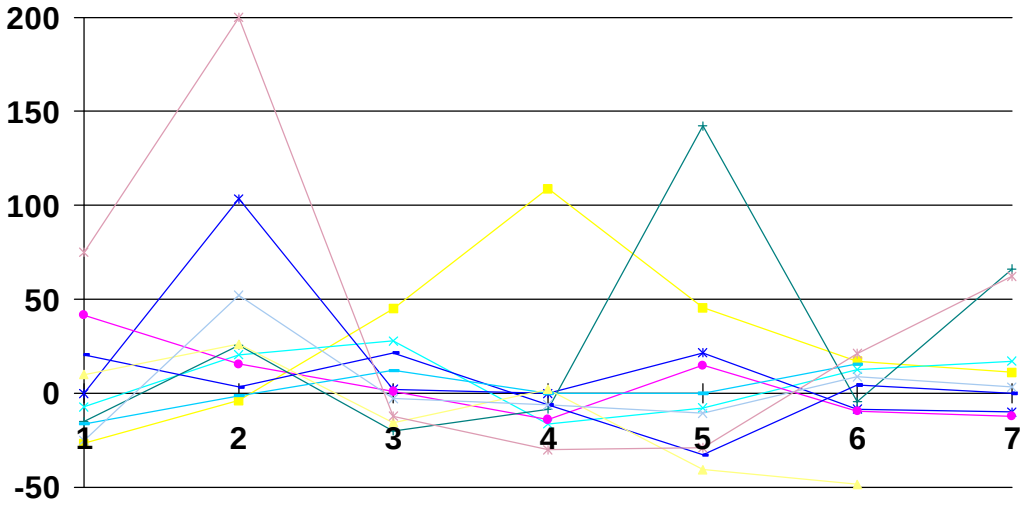
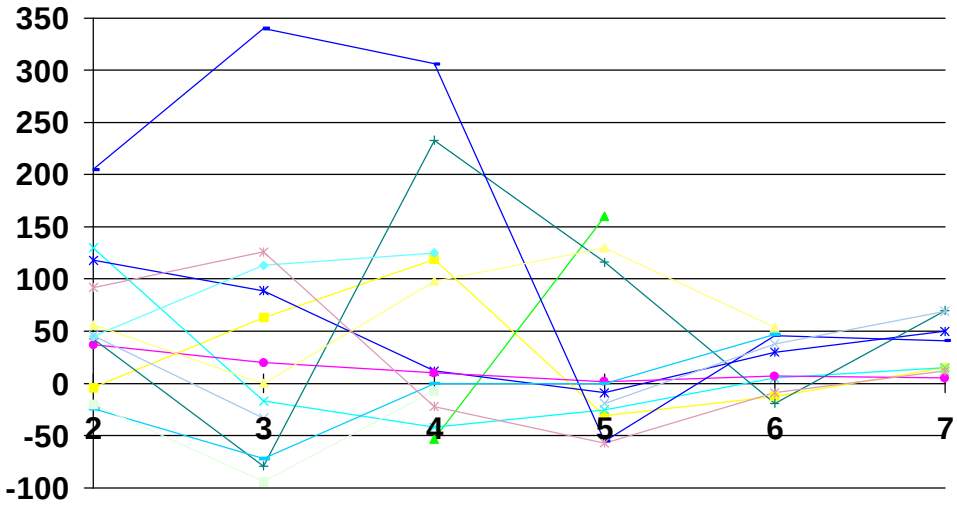


Fig. 4: Size Estimating Error%



Estimating in the PSP is based on predicting time from effort, and relating effort to program size. But the correlation between development time and LOC produced varies greatly. The clearest are C (Figure 5) and Visual Basic (Figure 6). But LOC count is not useful for Object-oriented languages (Figure 7). All 4GL developers stressed the need to work out other size metrics for their environments; and the need for tool support to track them.

Fig. 5: C time/size correlation=0.93

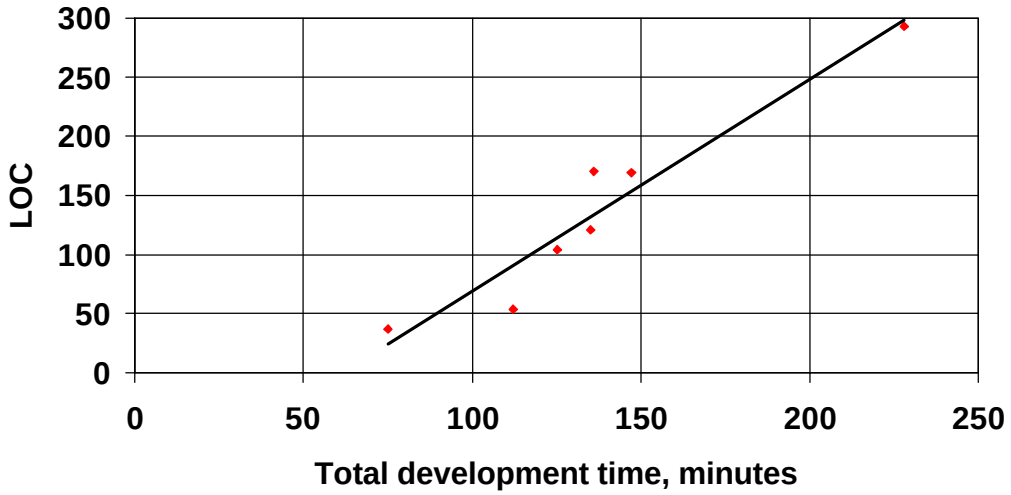


Fig. 6: VB time/size correlation=0.71

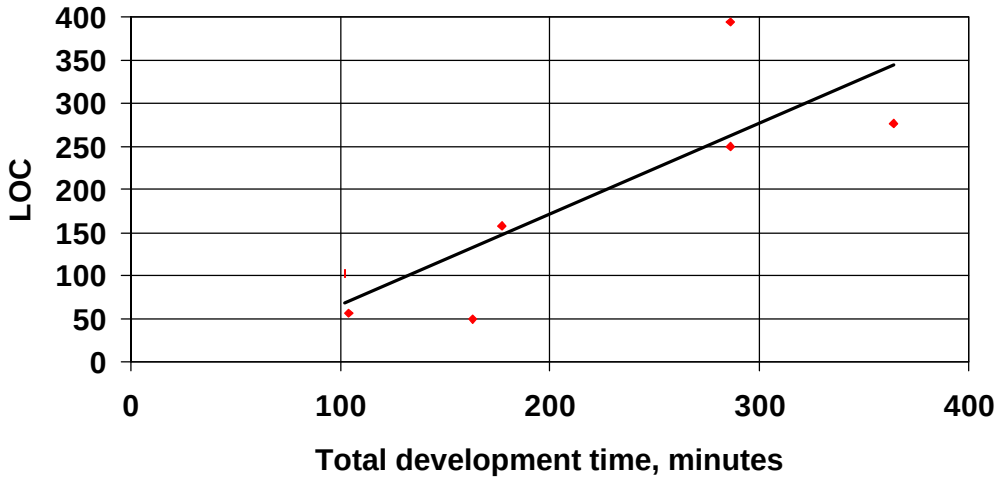
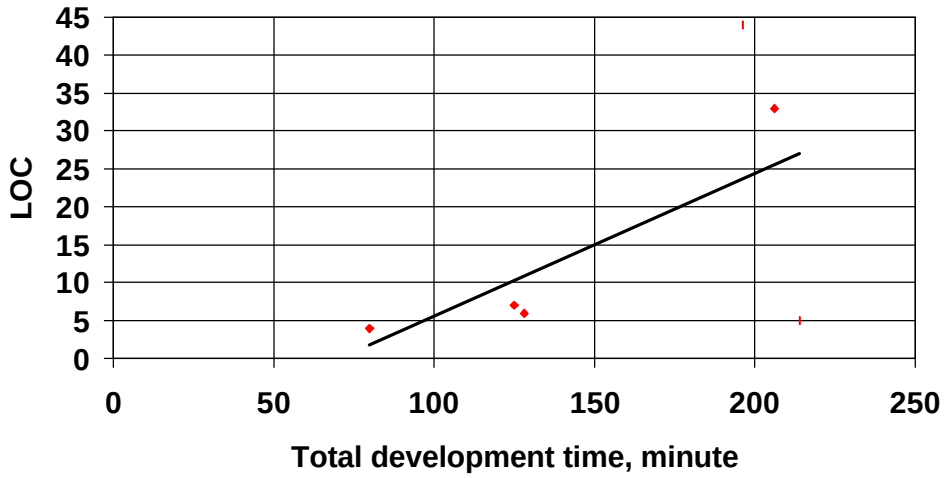


Fig. 7: ObjectStar time/size corr=0.34



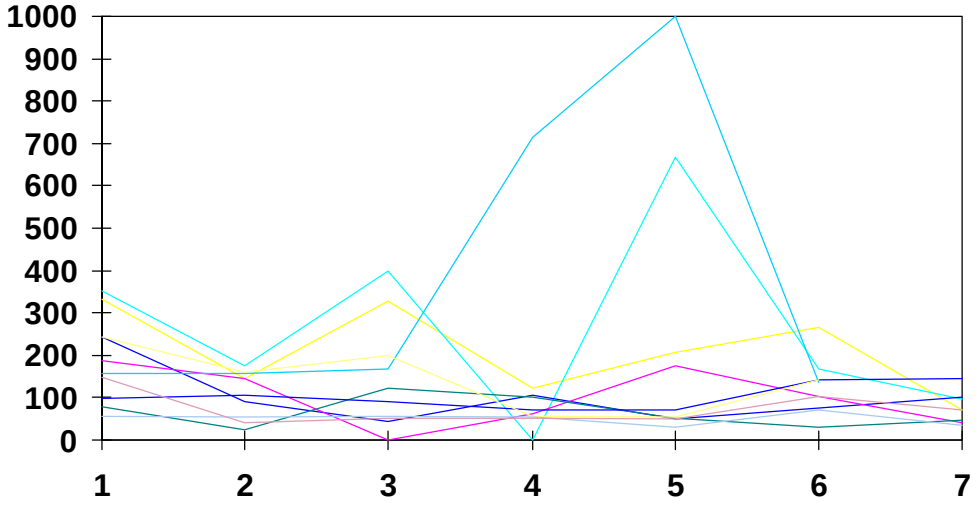
Defect reduction

There appears to be no significant correlation between years of experience, either in total or in the current language, and the defect rate (Figure 8).

Fig. 8: Defects/KLOC average all assignments

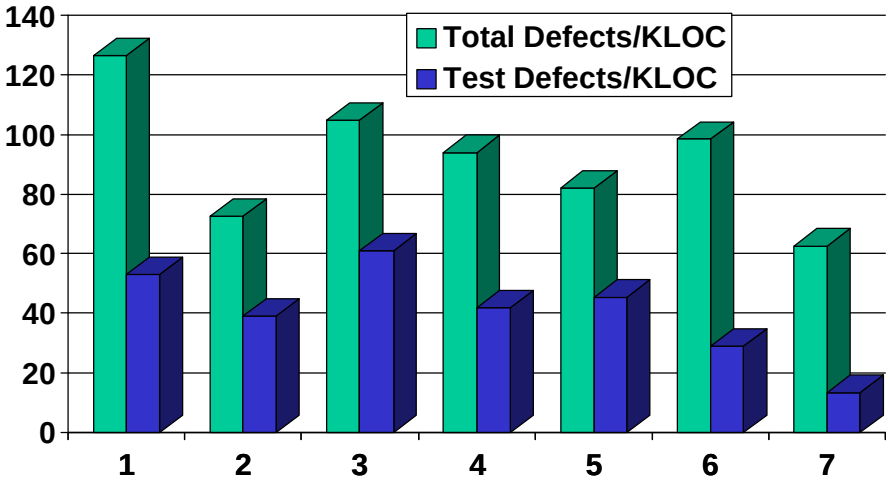
At the beginning of the course, their defect rates ranged from 25 to 350 defects per 1000 lines of code (KLOC) with an average of about 150. In the PSP, we count all defects including those found in desk checking, compile, and test. These figures are very similar to other reported PSP results.[2] The detailed results show much variability (Figure 9)

Fig. 9: Total Defects/KLOC
10 students, 7 assignments



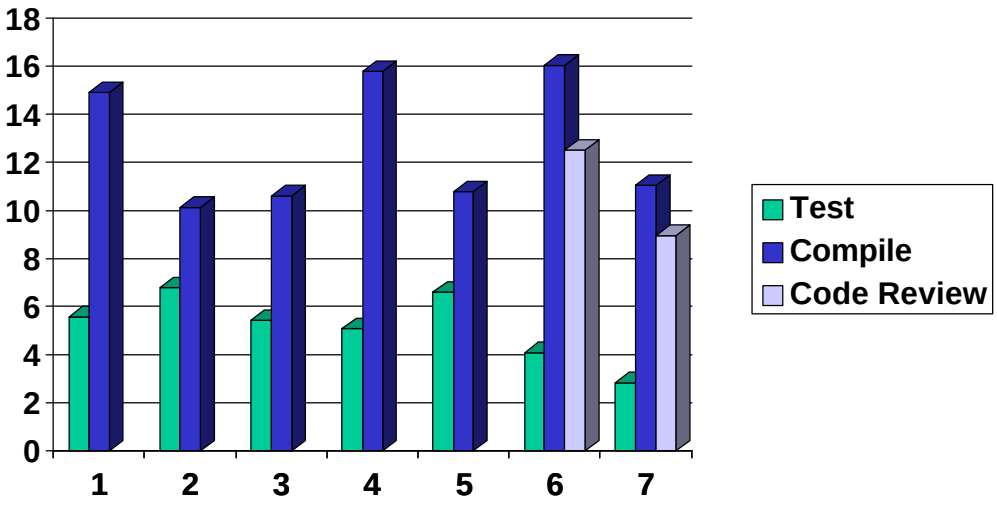
The peak at 1000 was one participant who wrote 4 LOC and had 4 defects.
Taking the weighted average shows the trend more clearly (Figure 10)

Fig. 10: Group average defect rate



A principal PSP measure is the defect removal rate. The average defect removal rate per hour for the participants were 11/hr for code review, 13/hr for compile, and only 6/hr for test. When defects are removed earlier, the test rate drops to about 3/hr (Figure 11)

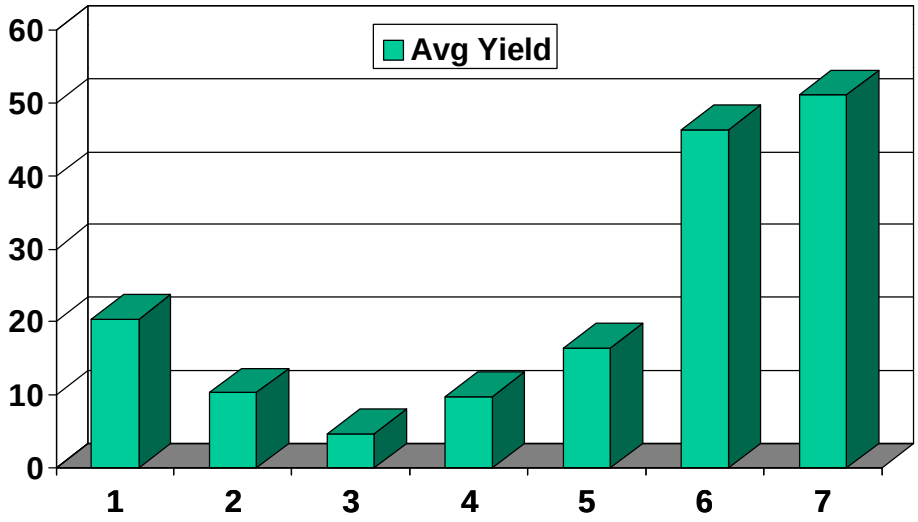
Fig. 11: Defects/hr removal rates



A key focus of the PSP is to push defect removal earlier in the process, to save overall development time; one objective of this is to prevent or remove defects before the first compile. As can be seen from the yield chart, the participants succeeded in an almost 4 times improvement, moving from an average of 13% to 48% yield when design & code reviews were introduced.

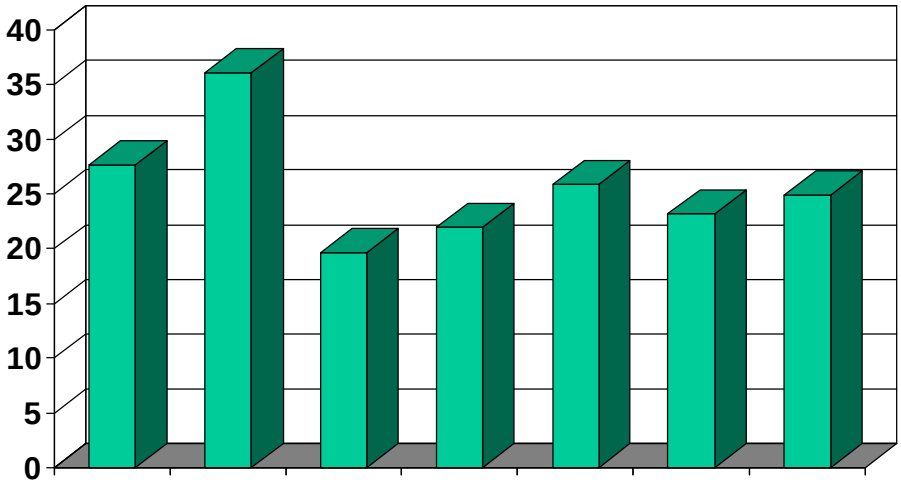
Fig. 12: Group average of Yield

= 100* Defects removed before compile
/ Defects injected before compile



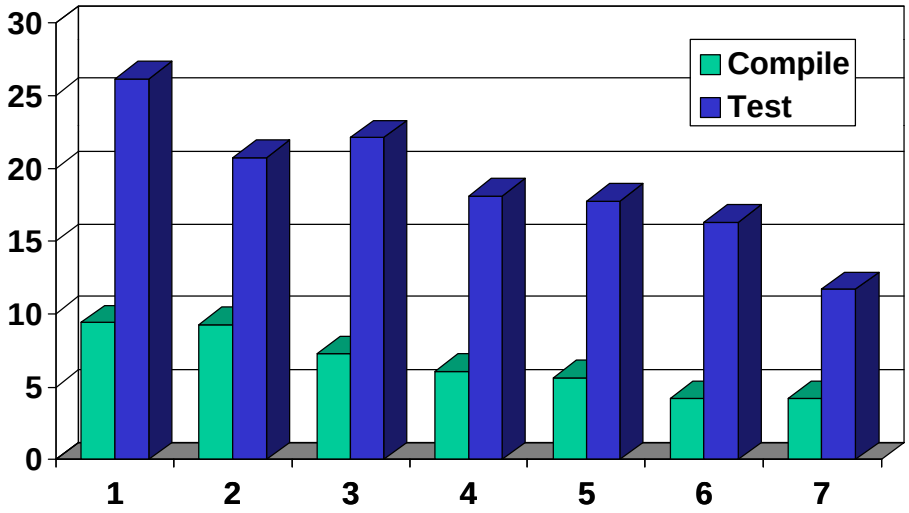
The improvement in yield was achieved at no loss of productivity - it stayed within 5LOC of an average of c. 26 LOC/hr (Figure 13)

Fig. 13: Group productivity LOC/hr



The result of this was a drop in the percentage of time spent in the compile phase from 9% to 4%. Incidentally, 9% is already about half of the amount a typical college student would spend in compile phase. The percentage of total time spend in the test phase declined from 26% to 12%.

Fig. 14: Compile & Test times as % of total development time



As only one of each of design & code review exercise was done, no conclusions are evident from the Yield charts. The intention is to find an optimum review rate to recover the best yield of defects. Code review finds defects on average twice as fast as testing, and almost as fast as the compiler.

No.	Code	Code Review	Compile	Test	Ratio CR/test
2	1.5	26.7	27.1	2.4	11.0
3	0.2	27.3	33.7	7.9	3.5
4	1.4	12.0	10.5	3.4	3.5
6	0.4	17.5	35.8	7.2	2.4
1	-	9.1	42.0	6.4	1.4
7	0.2	4.4	4.6	4.3	1.0
8	2.5	2.4	-	6.2	0.4
5	3.4		29.2	6.9	
9	1.6	-		12.0	-
10	0.5	-	3.5	2.2	-
Average	1.2	10.8	13.7	5.4	2.0

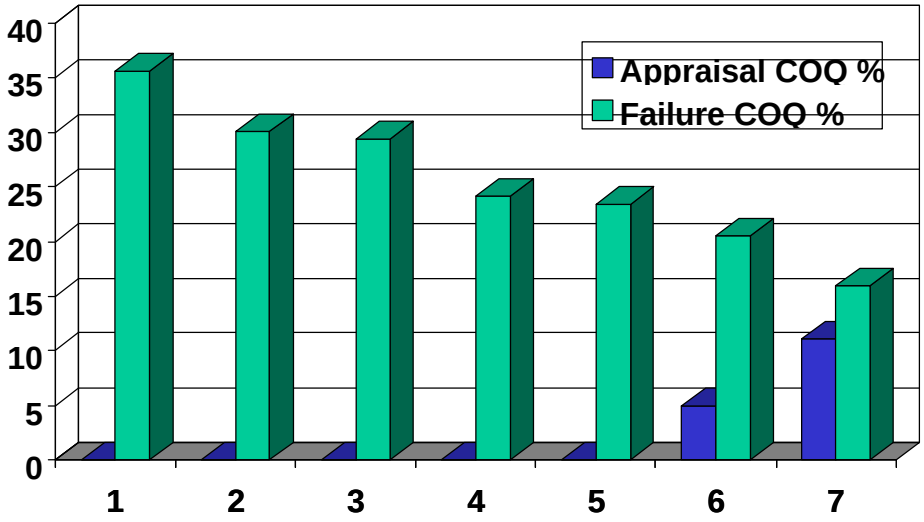
Table 4: Defect removal rate/hour in each phase

Overall, the group are reducing the amount of time spent handling quality failure, bringing it more into balance with the time on quality appraisal.

Fig. 15: Cost of Quality measures

Appraisal = (Des.Rev+Code Rev) / Total time

Failure = (Compile+Test) / Total time



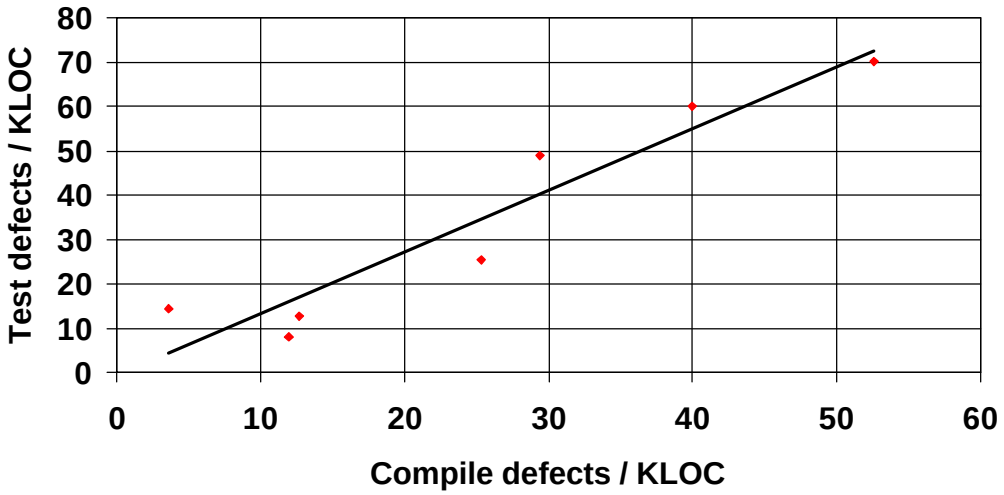
A common objection raised to counting compile defects is that they bear no relation to final defects; that the compiler should be used to catch syntax errors and these do not need to be tracked. Table 5 shows the correlations between the incidence of defects at Compile phase and test phase. Visual Basic (Fig. 16) and C++ both show high correlations. This tends to bear out the assertion that the more defective a product is at an early stage of the process, the more defective it will be at later stages. We do not have measures in the PSP of post-release defects, which are the most damaging to relations with the users and the expensive to fix. In industry, relating these to test defects should indicate how to achieve the minimum total cost of production and maintenance.

Language	Correlation
Visual Basic	0.90
C++	0.74
C++	0.73
C	0.71
C	0.61
SQLWindows	0.58
C	0.29
PL/1	0.12
C	0.01
C	0.00

Table 5

Fig.16: Test vs. Compile defects

(VB correlation=0.9)



4. Participants' Evaluations of the PSP

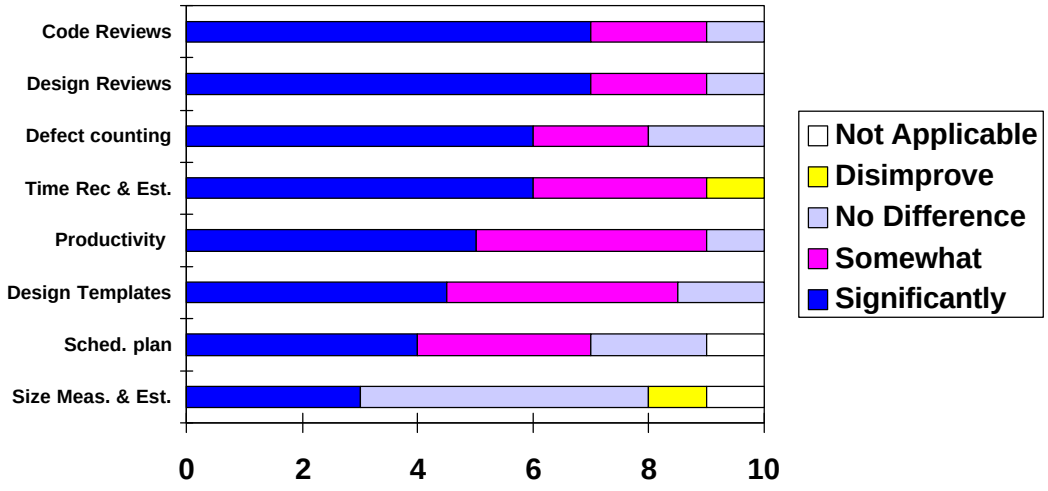
Analysis of a questionnaire at the end of the course.

Q1) What components of the PSP are you already putting into practice?
Design, Design Review. Code review. Time accounting. Defect Analysis.

Q2) What components do you think you will not use?
4 mentioned **LOC Counting in any form**

Q3) To what extent do you believe that use of the PSP techniques would improve your performance as a software engineer? (Figure 17 shows the responses as a chart)

Fig. 17: Survey of opinions "Effect of PSP on my process"



Details of their comments on PSP techniques:

PSP0 The Baseline process plus Time & Defect Recording.

"Even approximate recording is better than nothing."

Time recording was seen as a winner - at last they could show to management where the time was going. The amount of time consumed by interruptions was also an eye-opener.

Some already do defect counting & tracking in their companies. For those new to it, defect recording and analysis by category was another eye-opener. Later, this is seen as the foundation for review checklists and Process Improvement Proposals (PIPs).

"Reduce escalation costs of defect removal & quality"
"There seems to be a high correlation between compile and test defects."

PSP0.1 Time Estimation ; Program Size measurement & Productivity tracking

"I have found productivity rates very useful in the past to throw up problems ASAP"
"Not really part of my job"
Most people reported satisfaction with their estimation skills

PSP1 Program Size Estimation

"Some form of size estimation will give an idea of the work involved."
"Effort expended on GUI form design needs to be accounted for. "
"A lot of data and practice is needed before this method can be used with confidence."
All thought size estimation was hard work.

PSP1.1 Resource and Schedule Estimation & Planning

"Probably the most beneficial area of the PSP. Current scheduling is largely based on guesswork"
"Management set the schedule on time estimates."

PSP2 Design Reviews

"Good S.E. starts here."
"Ensures requirements are covered, interpreted and implemented as expected"
"Gain productivity by saving time by removing defects prior to test."
"We conducted a Design review and [...] defects were found by two different people. I have found [Review] benefit to be enormous not only to remove defects but for knowledge transfer."

"Very good to do but difficult to justify the time."
"Definite reduction in defects."
"Extremely useful for removing defects."
"Already carry out reviews"
"Eliminates more defects before test, better quality code that can be taken over by others."
"The use of code reviews has been proven to me to be a necessary evil. It has definitely been shown to have reduced time [...and...] multiple cost-inducing defects later on in the process."

Q4) To what extent do you believe more widespread application of PSP would benefit your organisation?

"Everyone would have to apply it to see real benefits. "
"The discipline and commitment to it is difficult "
"Better quality products, better quality people."

Q5) What are the main barriers you consider would have to be overcome to bring PSP into widespread use?

"Time taken to learn and implement the PSP; Old habits die hard!"
"Short term negative effect on productivity may not be acceptable. A slow introduction to PSP may give an even balance."
"Convincing experienced developers to follow all that PSP suggests."
"Change of attitudes, change of work practice, acceptance of change."
"None, we have a small development team (2) and both attended the cluster."
"Less emphasis (time-consuming) on paper/report filling - i.e. better automation."
"Non-paper-based means of tracking ; getting people to change their process to a more disciplined one."

Their conclusions and objectives from the course

The PSP is not a "magic bullet". The methods take time and effort to learn and they require consistent discipline to keep using. Not for nothing is the book called "A Discipline for Software Engineering". The following is a quotation from one participant's final report.

"I approached the PSP with a great deal of scepticism. What I did find useful was the production of visible hard facts about the way I conduct my own brand of software engineering. This can be seen (and most importantly appreciated by senior management) in the time-accounting and productivity reports. [PSP] requires a great deal of self-discipline and, at least at the start, a great 'leap of faith' on the part of the engineer."

It may be added that a greater amount of faith is required on the part of their manager, who (in the PSP course anyway) does not get access to all the 'hard facts' the engineer does.

Participants' Objectives for their work

- Faithfully keep a detailed account of my time and tasks
- Exercises were artificial - I'll start again collecting data on real projects.
- Identify which PSP level to implement (try PSP3), its costs, and apply it diligently.
- Apply PSP to build up a database for estimation.
- Design an accurate and relevant PROBE proxy for each language.
- Continue Design & Code Reviews.
- Design full PSP spreadsheet.
- Search for utilities to automate the process.
- Reduce Unit Test defects to under 100/KLOC within the next 12 months.
- Take more care when coding
- Spend more time designing and reviewing programs. Also get someone else to review the design each time before coding.
- Keep records of performance and review performance for each new project.

Our conclusions for future courses

To help the transition to organisational adoption, we included extra items on the course : an exercise in two-person peer review, and a presentation by Joc Sanders on "People issues in Culture change in organisations". Both were well received. On a long course like this, experienced developers value the dialogue and exchange of ideas with their peers. We consider that future courses should concentrate on those areas most positively received above - defect reduction and reviews - and include sessions on practical software improvement techniques which are not currently part of the academic PSP.

Where do we go next?

Research will be ongoing in PSP techniques related to:

1. Requirements specification and RAD techniques such as timeboxing.
2. Maintenance involving support calls for defects found in use can be monitored in the same way. This is where many companies begin their effort at software improvement, to save wasting money in bug fixes and spend it on customer satisfaction instead.
3. Documentation development; WSH himself devised a process for writing the PSP book and devotes some space to writing about it.
4. Object orientation : the 3GL LOC focus of the PSP course does not suit Object-oriented or component-based development. Participants could not, in the time available, change the PSP method during the course, but we are hoping they will be able to report on how they adapt it to their OO environments.
5. Automated time recording - the author uses a time recording tool to track tasks down to the minute. It's a generic package, one of many on the market, and some have proposed specific PSP time-recording utilities that run in the background.
6. Automated size & effort metrics: The tools posted on the psp-users mail list are typically either Unix or DOS tools that rely on traditional source code text files. The author will be investigating using the data dictionary aspect of popular tools such as Microsoft Access to assist in the automated tracking of work.
7. Both of the above, plus defect measurement, really need to be integrated into the developer's IDE to make it easy to use; otherwise the temptation is to forgo the paper-filling when the pressure comes on. As Ed Yourdon says, "There is no point in using a method that cannot be trusted in emergencies". The SEI has a paper on its Web site containing a specification for automated support for the PSP. All we need is a market large enough to make it worthwhile!
8. How to work on issues of disclosure of data to management; the relationship to an organisational metrics programme.
9. Tackling commitment, given programmers' reluctance to perform overhead data recording activities.
10. A more eclectic course will be devised to tailor "best practices" at an individual level to the needs of industry. Humphrey has a second PSP book [4] also aimed at students which is less heavily statistical and may suit practicing software engineers with some adaptation.

References

[1] "A Discipline for Software Engineering" by Watts S. Humphrey, Addison-Wesley 1995.
[2] Published results of Andrew Worsley on the Web: http://www.cm.deakin.edu.au/~peter/PSP_data/Contributor_index.html
[3] Implementing Concepts from the Personal Software Process in an Industrial Setting K. El Emam, B. Shostak, and N. H. Madhavji Proceedings of the 4th International Conference on the Software Process, pages 117-130, IEEE CS Press, 1996.
[4] "Introduction to the Personal Software Process" by Watts S. Humphrey, Addison-Wesley 1996.