C. Wohlin, "The Personal Software Process as a Context for Empirical Studies", IEEE TCSE Software Process Newsletter, pp. 7-12, No. 12, Spring 1998.

# The Personal Software Process[1] as a Context for Empirical Studies

**Claes Wohlin**
**Dept. of Communication Systems, Lund University**
**PO Box 118, SE-221 00 Lund, Sweden**
**E-mail: claesw@tts.lth.se**

## Abstract

This paper discusses the use of the Personal Software Process (PSP) as a context for doing empirical studies. It is argued that the PSP provides an interesting environment for doing empirical studies. In particular, if we already teach or use the PSP then it could be wise to also conduct empirical studies as part of that effort. The objective of this paper is to present the idea and discuss the opportunities in combining the PSP with empirical studies. Two empirical studies, one experiment and one case study, are presented to illustrate the idea. It is concluded that we obtain some new and interesting opportunities, in particular we obtain a well-defined context and hence ease replication of the empirical studies considerably.

## 1. Introduction

Different decades have different trends in software engineering. In the 90s, we have seen a strong focus on the process and also on the use of empirical methods in software engineering. The need for a scientific approach to software engineering has been stressed, see for example [Fenton94]. In this paper, we would like to look at the opportunity to combine two of the trends in the 90s, i.e. process focus and empirical studies.

The objective of the paper is to highlight and discuss the opportunities of performing empirical studies within the context of the Personal Software Process (PSP). The PSP is a well-defined process and the process in publicly available through the book by Humphrey [Humphrey95]. This means that studies conducted within this context rather easily can be replicated, which is critical to the success of applying empirical methods in software engineering.

The paper is organized as follows. In Section 2, the PSP is briefly discussed. Section 3 provides a brief introduction to empirical studies. Section 4 discusses the ability to use the PSP as a context for empirical studies. The advantages, challenges and opportunities are outlined. In Section 5, two example of studies conducted with the PSP as an empirical context is presented to illustrate the approach. Finally, in Section 6, a brief discussion is provided.

## 2. The Personal Software Process

[1] The Personal Software Process and PSP are service marks of Carnegie Mellon University.

The Personal Software Process (PSP) has gained lot of attention since it became publicly available [Humphrey95]. The objective of the PSP is basically to provide a structured and systematic way for individuals to control and improve their way of developing software. We have seen papers, for example [Humphrey96], presenting the outcome of the PSP, primarily from students taking the PSP as a course. The PSP is currently used in a number of universities and industry is also becoming interested in applying the PSP.

At Lund University, we run the PSP as an optional course for students in the Computer Science and Technology program and the Electrical Engineering program. Most students take the course in their fourth year, and the course is taken by 50-70 students. The course is run for the second time during the autumn of 1997. The main objective of the course is to teach the students the use of planning, measurement, estimation, postmortem analysis and systematic reuse of experiences. It is from a course perspective more important to teach the students the techniques packaged within the PSP than actually teaching them the PSP for future use.

## 3. Empirical research

Another area which attracts attention is the use of empirical methods in software engineering. The need for experimentation was stressed already in the 1980s [Basili86], but it is during the last couple of years that we have seen a stronger focus on the use of empirical methods, for example, emphasized in [Fenton94]. Experiments and case studies will allow us to gain a better understanding of relationships in software engineering and they will also allow us to evaluate different hypotheses. Numerous books on design and analysis of experiments in general are available, for example, [Montgomery97], and case studies are, for example, discussed in [Stake95].

One major difficulty in empirical studies is the validity of the results. In other words how do we interpret the results and what conclusions can we draw? A particular problem is, of course, to find suitable subjects (participants in the experiment). It is desirable to use industrial software engineers, but many times this is unfeasible. A suitable starting point is therefore many times to start with experiments at the universities using students as subjects, i.e. the experiments are conducted in an educational context. The use of students as subjects is, of course, a major threat to the validity, but on the other hand it could be good to start with an experiment in a university setting and based on the outcome we can, as part of a technology transfer process, replicate the experiment in industry and then continue with a pilot project.

Experiments are regularly conducted with students as subjects, see for example [Briand97]. The experiment can be done as part of a course in software engineering or as a separate activity where the students are attracted to the experiment based on some reward. Thus, we often have to resort to using the students and hence we would like to raise the question: is it possible to use the PSP as a context for empirical studies, including both experiments and case studies? If we use the PSP in the education could we also use the PSP course as a means for empirical studies?

## 4. The PSP and Empirical Studies

The PSP can be viewed from two different perspectives when it comes to empirical studies. First, it is important to evaluate the effect of the PSP. This means that the PSP is the object of the study. We have seen results published [Humphrey96], but further studies are needed. This includes both reports on the outcome from taking the PSP as a course and from industrial use of the PSP. It is, however, not the intention here to discuss this matter. Second, assuming that we have started to teach and use the PSP, the PSP can be used as a context for empirical studies and hence as a vehicle for evaluating different methods and techniques, and to study different relationships in software engineering.

The second issue is the main objective of this paper. In particular, the objective is to highlight the opportunities and limitations of using the PSP as a context for empirical studies. Two different types of empirical studies can be identified:

- Experiments aimed at testing a hypothesis, for example, comparing two different methods for inspections. For this type of studies we apply methods for statistical inference, and we would like to show with statistical significance that one method is better than the other [Montgomery97].
- Case studies used to build models relating attributes to each other, for example, prediction models. One example of a prediction model may be to predict the number of faults in testing based on the number of defects found in compilation and the time spent in inspections. We mainly apply multivariate statistical analysis in this type of studies. The analysis methods include linear regression and principal component analysis [Manly94].

Both these types of studies are briefly illustrated within the context of the PSP below. It should be noted that the main objective is to illustrate empirical studies using the PSP rather than presenting the studies in detail.

A key question to consider is, of course, why should we use the PSP as a context for empirical studies? Or we could negate the question, and ask why should we not use the PSP as a context for empirical studies? We would like to argue, since we often have to resort to using students in experiments anyway, that it is suitable to use the PSP (at least if we teach the PSP independent of our interest in experimentation). The PSP provides a context which includes collection of several measures which are valuable when experimenting. Moreover, we also believe that the PSP can be the starting point for case studies, i.e. before going out in industry and perform a major study, we can gain a first (and valuable) insight by studying the PSP and its outcome. Thus, the advantages, challenges in using the PSP, and opportunities are:

Advantages:

- **Context**. The context is given by the definition of the PSP as described by Humphrey [Humphrey95]. We may want to change the proposed PSP slightly, but basically the context is provided, and hence we do not have to define the context and describe it very carefully to allow for others to understand our study from the context perspective.
- **Replication**. The context also forms the basis for replication. A major problem in empirical studies is that in order to come up with generally valid observations, we must be able to perform an study several times to build up general experience. Thus, the PSP may be one way to ease replication. In other words, experiments and case studies can be conducted at several

places using the PSP simultaneously. The PSP provides a stable process and the process description is generally available.
- **Measures**. This is also closely related to the PSP. Measures are collected as an integrated part of the PSP, and it is fairly easy to add measures of specific interest for an empirical study. Thus, the PSP provides a good starting point for collecting measures to use for hypothesis testing of model building.

Challenges:
- **Scaling**. The PSP implements activities performed in large-scale project, hence scaling down, for example, planning and estimation to the individual level. A major challenge in empirical studies is to generalize, see also validity, the observations. The major challenge in using the PSP as context is the ability to scale the observation to other environments, and in particular to large-scale software development. On the one hand, it is difficult to scale individual results to large project, on the other hand the PSP is supposed to act as a down-scaled project.
- **Validity**. The validity of the observations and findings is crucial. We would like to be able to generalize the observations. In order to do this, we must consider different types of validity, for example internal and external validity, [Cook79]. The actual validity for different studies must be addressed separately, as the ability is highly dependent on the study and what we intend to generalize.

Opportunities:
The PSP provides some opportunities for empirical studies. We may study the use of different techniques and methods, or investigate the relationships between different attributes. The main limitation of using the PSP as a basis for empirical studies is that we cannot use it to study group activities. It is possible to experiment with using different reading techniques on an individual basis, but we are unable to study the use of inspections and group meetings. Thus, we cannot expect to use the PSP as a context for experimentation for all types of empirical studies, but it is our firm belief that it opens some new opportunities and that the major inhibiting factor is our imagination.

It is clear that the PSP provides opportunities for empirical studies. In the following section, we will show two examples of studies conducted with the PSP as a context. The main objective of the examples is to illustrate the use of the PSP as context, and not to provide any deep insight into the actual empirical studies.

## 5. Illustration of using the PSP for Empirical Studies

### *5.1 Introduction*

The main objective of this section is to illustrate the use of the PSP as a context for empirical studies. The actual results of the studies are presented here, but not in full detail.

The objectives of the empirical studies are:
- to evaluate the difference in fault density based on prior experience of the programming language, and
- to investigate the relationships between different performance measures.

For the first case, we have collected background information on the experience of the programming language used. In our particular case, the students should use C as programming language independent of their prior experience. In the second case, we decided to formulate seven performance measures which we derived for all students. The objective is to investigate what dimensions we are able to measure. On a general level, we are normally interested in the following attributes: quality, productivity, cycle time and predictability. The quality is sometimes, for reasons of simplicity, measured in terms of fault content.

### *5.2 Context*

The Empirical study is run within the context of the PSP. Moreover, the study is conducted within a PSP course given at the Department of Communication Systems, Lund University, Sweden. The course was given in 1996-97, and the main difference from the PSP as presented in [Humphrey95] is that we provided a coding standard and a line counting standard. Moreover, the course was run with C as a mandatory programming language independent of the background of the students. The study is focused on the outcome of PSP. The PSP course is taken by a large number of individuals (this particular year, we had 65 students finishing the course). Thus, we have 65 participants (subjects) in the study. The experiment can be regarded as a quasi-experiment since students signed up for the course and hence we lack randomization [Daly97]. In a case study, we do not expect to have randomization. In general, we have much less control in a case study than in an experiment.

### *5.3 Planning*

As part of the first lecture, the students were asked to fill out a survey regarding their background in terms of experiences from issues related to the course, for example, knowledge in C. The students were required to use C in the course independently of their prior experience of the language. Thus, we did not require that the students had taken a C-course prior to entering the PSP course, which meant that some students learnt C within the PSP course. This is not according to the recommendation by Humphrey, see [Humphrey95]. The hypothesis of the experiment based on the C experience was that students with more experience in C would make fewer faults per lines of code.

The case study is based on investigating several performance measures and after the course evaluate if they measure several different dimensions, and in particular if we are able to capture quality, productivity, cycle time and predictability from the performance measures. It should be noted that we use all 10 programming tasks in the PSP course as the basis for determining the performance. The following measures were defined as performance measures: Total number of faults, Fault density, Program size, Development time, Productivity, Predictability of size, and Predictability of time. The objective of the second study does not require anything in particular during the course, since it is primarily an analysis at the end.

For the first case, we have the following hypothesis:
Null hypothesis, H0: There is no difference between the students in terms of number of faults per KLOC (1000 lines of code) based on the prior knowledge in C.

- H0: Number of faults per KLOC is independent of C experience.
- H1: Number of faults per KLOC changes with C experience.

Measures needed: C experience and Faults/KLOC.

The C experience is measured by introducing a classification into four classes based on prior experience of C (ordinal scale). The classes are:

1. No prior experience.
2. Read a book or followed a course.
3. Some industrial experience (less than 6 months).
4. Industrial experience.

The second investigation requires that the following data are collected: Program size (estimate and actual), Development time (estimate and actual), and Number of faults. From these measures, we are able to derive the performance measures. The fault density of Faults/KLOC is also used in the first case. It should be noted that we are unable to evaluate the cycle time as we have no measures which capture the cycle time. This is difficult to achieve within the PSP. The best we can do is probably to measure delivery precision, primarily in terms of the number of late deliveries. We have, however, not kept track of this information or at least to the degree that we trust the data.

The experimental design for language experience is: one factor with more than two treatments. The factor is the experience in C, and we have four treatments, see the experience grading above. The dependent variable is measured on a ratio scale and we can use a parametric test for this hypothesis. The ANOVA test is hence suitable to use for evaluation.

For the case study, we would like to use principal component analysis to study what dimensions we are capturing with our seven performance measures. It is quite common that we collect a large number of measures, but basically we are only capturing a few dimensions due to multicollinearity between the different measures.

### *5.4 Validity evaluation*

This is a difficult area. In our particular case, we have several levels of validity to consider. Internal validity can be divided into: within the course this year, and between years. External validity can be divided into: students at Lund University (or more realistically to students from programs taking the PSP course), the PSP in general, and for software development in general.

The internal validity within the course is probably not a problem. The large number of tests (equal to the number of students) ensures that we have a good internal validity, probably both within the course this year and similar results can be expected in the future, if running the course in a similar way.

Concerning the threats to the external validity, it is difficult to generalize the results to other students, i.e. students not taking the course. They are probably not as interested in software development and hence they come from a different population. The results from the analysis can probably be generalized to other PSP courses, where it is feasible to compare participants based on their background in terms of computer science or electrical engineering or experience of a particular programming language.

The results are found for the PSP, but they are likely to hold for software development in general. This is motivated by the following observations for the two studies:

- There is no reason that people having different background experience from a particular programming language perform differently between the PSP and software development in general. Thus, for the language experience, we expect similar results in other environments.
- The performance measures can be collected for other environments than the PSP, and there is no reason that we should not get a similar grouping of the different measures. Thus, we believe that the results can be generalized to other contexts.

## *5.5 Operation*

The subjects (students) are not aware of what we intend to study. They were informed that we wanted to study the outcome of the PSP course in comparison with the background of the participants. They were, however, not aware of the actual hypotheses stated. The students, from their point of view, do not primarily participate in an empirical study; they are taking a course. All students are guaranteed anonymity.

The survey material is prepared in advance. Most of the other material is, however, provided through the PSP book [Humphrey95]. The empirical study is executed over 14 weeks, where the 10 programming assignments are handed in regularly. The data are primarily collected through forms. Interviews are used at the end of the course, primarily to evaluate the course and the PSP as such.

## *5.6 Data validation*

Data were collected for 65 students. After the course, the achievements of the students were discussed among the people involved in the course. Data from six students were removed, due to that the data were regarded as invalid or at least questionable. Students have not been removed from the evaluation based on the actual figures, but due to our trust in the delivered data. The six students were removed due to:

- Data from two students were not filled in properly.
- One student finished the course much later than the rest, and he had a long period where he did not work with the PSP. This may have affected the data.
- The data from two students were removed based on that they delivered their assignments late and required considerably more support than the other students, hence it was judged that the extra advice may have affected their data.
- Finally, one student was removed based on that his background is completely different than the others.

This means removing six students out of the 65, hence leaving 59 students for statistical analysis and interpretation of the results.

## 5.7 *Analysis and interpretation*

### 5.7.1 Analysis of experiment

For the experiment, we use descriptive statistics to visualize the data collected. From plotting the data, it is obvious that we have one outlier. If we look at the data including the outlier in the analysis, it seems as there is a weak tendency (when looking at the mean value) towards that more experience means lower fault density. If we remove the outlier, the tendency is still there although very weak. The data are summarized in Table 1. Thus, we do not expect to find any support for the hypothesis that language experience affect the fault density.

**TABLE 1:** Faults/KLOC for the different C experience classes.

| Class | Number of students | Median value of Faults/KLOC | Mean value of Faults/KLOC | Standard deviation of Faults/KLOC |
|---|---|---|---|---|
| 1 | 31 | 66.0 | 72.7 | 29.0 |
| 2 | 19 | 69.7 | 68.0 | 22.9 |
| 3 | 6 | 63.6 | 67.6 | 20.6 |
| 4 | 2 | 63.0 | 63.0 | 17.3 |

The next step is to apply an ANOVA test to evaluate the hypothesis that more experience in C means fewer faults/KLOC. The results of the analysis are shown in Table 2.

**TABLE 2:** Results from the ANOVA test.

| C experience vs. Faults/KLOC | Degrees of freedom | Sum of squares | Mean square | F-value | p-value |
|---|---|---|---|---|---|
| Between groups | 3 | 3483 | 1161 | 0.442 | 0.724 |
| Within groups | 55 | 144304 | 2624 | | |

As expected, the results from the analysis are not significant. Thus, we are unable to show that there is a significant difference in terms of number of faults/KLOC based on C experience. Since the number of students in class 3 and 4 is very limited, class 2, 3 and 4 are grouped together to study the difference between class 1 and a grouping of class 2-4. A t-test was performed to evaluate if it was possible to differentiate between class 1 and the rest. No significant results were obtained.

### 5.7.2 Analysis of case study data

For the case study, we are interested to investigate if we are able to capture different dimensions through our seven performance measures. In particular, we would like to see how many dimensions our seven measures actually capture. In order to do this, we apply a principal component analysis. The results of the analysis are presented in Table 3.

**TABLE 3:** Results from the principal component analysis.

| **Performance measure** | **Factor 1: Faults** | **Factor 2: Productivity** | **Factor 3: Predictability** |
|---|---|---|---|
| Faults | 0.869 | 0.395 | 0.019 |
| Faults/KLOC | 0.886 | 0.183 | 0.177 |
| Development time | 0.815 | -0.282 | -0.083 |
| Program size | 0.156 | 0.824 | 0.064 |
| Productivity | -0.570 | 0.778 | 0.134 |
| Predictability Size | -0.218 | -0.141 | 0.740 |
| Predictability Time | -0.036 | -0.247 | 0.740 |

From Table 3, we note that the seven measures can be grouped into three factors. The first factor seems to mainly capture faults. The development time is included in this factor, which may be regarded as a surprise, but it can be explained by that a driving time factor is the number of faults. People who have many faults take longer time to develop their programs, hence supporting the hypothesis that fault prevention and early fault detection is important for the development time. The second factor includes program size and productivity. This result indicates the difficulty we have in capturing productivity, i.e. people who write large programs seems to have a higher productivity. In this case when the students develop the same programs, productivity should be measured in terms of time to implement the functionality rather than as defined here. The third factor clearly captures the ability to estimate accurately, i.e. predictability.

From the case study, we can see that we manage to differentiate between the three factors: quality (in terms of faults), productivity (although a questionable measure) and predictability. The fourth major factor (see Section 5.3), i.e. cycle time, is not measured and hence, of course, not visible among the factors found in the principal component analysis.

## 6. Discussion

We would like to see the Personal Software Process as an opportunity for empirical studies, in addition to its original objective. Empirical studies are an important means to further understand, evaluate and improve software development. The empirical studies are often conducted in a student setting, and if we teach the PSP it may be combined with our need for empirical studies. Furthermore, it may be an important step in technology transfer. Experiments and case studies can be conducted in a controlled environment, before transferring the results to an industrial environment for further studies and implementation in the industrial development processes. It should also be noted that the empirical studies can be replicated easily based on the well-defined context provided by the PSP, hence providing a good basis technology transfer decisions.

The results from our two studies, briefly outlined in the paper, are interesting in themselves. The experiment shows that the experience of C programming does not influence the fault density. The case study illustrates that although trying to define seven different performance measures, we have actually only captured three main factors. It is also interesting to note that the factors are possible to separate, and that they are in accordance with the expectations.

We have already conducted several studies using the PSP as a context for empirical studies, and based on our experience and the results obtained we will continue our research in this direction. The studies so far include: two effort estimation studies (prestudy and main study) [Höst97, Höst98a], a programming language comparison study [Höst98b] and a study of the performance in the PSP based on the individual background [Wohlin98].

Finally, we would like to encourage others to perform empirical studies within the PSP both to replicate our studies and to perform other studies which could enlighten and improve our understanding of the underlying phenomena in software engineering.

## Acknowledgment

I would like to thank Per Runeson, Dept. of Communication Systems for valuable comments on a draft of this paper.

## References

[Basili86] V.R. Basili, R.W. Selby and D.H. Hutchens, “Experimentation in Software Engineering”, IEEE Transactions on Software Engineering, Vol. 12, No. 7, pp. 733-743, 1986.

[Briand97] L. Briand, C. Bunse, J, Daly and C. Differding, “An Experimental Comparison of the Maintainability of Object-Oriented and Structured Design Documents”, Empirical Software Engineering: An International Journal, Vol. 2, No. 3, pp. 291-312, 1997.

[Cook79] T.D. Cook and D.T. Campbell, “Quasi-Experimentation – Design and Analysis Issues for Field Settings”, Houghton Mifflin Company, 1979.

[Daly97] J, Daly, K. El Emam and J. Miller, “Multi-Method Research in Software Engineering”, Proceedings 2nd International Workshop on Empirical Studies of Software Maintenance, WESS’97, Bari, Italy, pp. 3-10, 1997.

[Fenton94] N. Fenton, S.L. Pfleeger and R. Glass, “Science and Substance: A Challenge to Software Engineers”, IEEE Software, pp. 86-95, July, 1994.

[Humphrey95] W.S. Humphrey, “A Discipline of Software Engineering”, Addison-Wesley, 1995.

[Humphrey96] W.S. Humphrey, “Using a Defined and Measured Personal Software Process”, IEEE Software, pp. 77-88, May 1996.

[Humphrey97] W.S. Humphrey, “Introduction to the Personal Software Process”, Addison-Wesley, 1997.

[Höst97] M. Höst and C. Wohlin, “A Subjective Effort Estimation Experiment”, Journal of Information and Software Technology, Vol. 39, No. 11, pp. 755-762, 1997.

[Höst98a] M. Höst and C. Wohlin, “An Experimental Study of the Individual Subjective Effort Estimation and Combinations of the Estimates”, Proceedings 20th International Conference on Software Engineering, Kyoto, Japan, April 1998 (to appear).

[Höst98b] M. Höst and C. Wohlin, “A Comparison of Programming Languages within the Personal Software Process”, Submission to Empirical Assessment & Evaluation in Software Engineering, EASE’98, Keele University, Keele, UK, March 1998.

[Manly94] B.F.J. Manly, “Multivariate Statistical Methods: A Primer”, Chapman & Hall, 1994.

[Montgomery97] D.C. Montgomery, “Design and Analysis of Experiments”, 4th edition, John Wiley & Sons, 1997.

[Stake95] R.E. Stake, “The Art of Case Study Research”, SAGE Publications, 1995.

[Wohlin98] C. Wohlin, A. Wesslén, M.C. Ohlsson, M. Höst, B. Regnell and P. Runeson, “A Quantitative Evaluation of the Differences in Individual Performance within the Personal Software Process”, Technical report, Dept. of Communication Systems, Lund University, 1998 (in preparation).