C. Wohlin and A. Wesslén, "Understanding Software Defect Detection in the Personal Software Process", Proceedings IEEE 8th International Symposium on Software Reliability Engineering, pp. 49-58, Paderborn, Germany, 1998.

Understanding Software Defect Detection in the Personal Software Process

Claes Wohlin and Anders Wesslén
Department of Communication Systems
Lund University
Box 118, SE-221 00 Lund, Sweden
E-mail: (claes.wohlin, anders.wesslen)@tts.lth.se

Abstract

There is a general need to understand software defects and our ability to detect defects during different activities. This is particularly important in relation to software process improvement, where one objective may be to decrease the number of defects. The Personal Software Process (PSP)[1] has gained attention during the last couple of years as a way to individual improvement in software development. Thus, the PSP is an interesting starting point to understand software defects and in particular the detection process. This paper presents a study of software defect detection for 59 students taking a PSP course. In summary, the study provides valuable insight into software defect detection in the PSP. Some of the results are interesting not only for the PSP, but from a general perspective in our understanding of software defect detection. The understanding of software defects forms the basis for improving the reliability of software.

1. Introduction

Software failures are costly. Reports regarding software problems are published regularly, ranging from minor issues to the year 2000 problem. Thus, a better understanding of software defects, their causes and possible improvements in the area are essential. Software failures are related to the reliability of the software.

Reliability is an external attribute. A program has a certain reliability from the perspective of the user. The internal attributes related to reliability are defects or faults. Defects are used throughout this paper when referring to the internal attribute. To improve the understanding of software reliability, an understanding of software defects is required. The latter includes all aspects related to defects, for example, introduction and detection. The main focus of this paper is the detection process.

This paper focuses in particular on the understanding of the defect detection process when using the Personal Software Process (PSP) [Humhprey95]. The objective is to study detection within the PSP, but also to use that understanding and experience as a basis for a more general understanding of software defect detection.

The paper is organised as follows. In Section 2, the PSP and the use of the PSP for empirical studies are discussed. Section 3 discusses the case study in terms of context and background. The data analysis is presented in Section 4, and Section 5 discusses the findings and future work.

2. The Personal Software Process and empirical studies

2.1. The Personal Software Process

The Personal Software Process (PSP) has gained lots of attention since it became publicly available [4]. 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 [5, 3], presenting the outcome of the PSP, both from students and industrial software engineers 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 Engineer-

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

ing program and the Electrical Engineering program. Most students take the course in their fourth year, and the course is taken by 40-70 students. The course is run for the second time during 1997-98. 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.

The PSP includes seven incremental steps in which the personal software process is gradually improved. The seven increments include four main increments denoted by PSP0, PSP1, PSP2 and PSP3. The main differences between these are:

- PSP0 - PSP1: Improved estimation techniques
- PSP1 - PSP2: Introduction of reviews
- PSP2 - PSP3: Incremental development is introduced

More information concerning the PSP can be obtained from [4, 6]. From a software defect detection perspective the major change is between PSP1 and PSP2, although we expect improvement regarding the number of defects between all the different steps, since the software process is improved.

In [4], the PSP is presented as a course, and this has been the basis in the course at Lund University. The basic course includes 10 programming assignments in which the PSP is presented and incrementally introduced. Furthermore, the course includes five reports. Three of the assignments use PSP0, PSP1 and PSP2 respectively, and the last assignment is done using PSP3.

2.2. Empirical studies

The use of empirical studies is emphasised by many researchers [1, 2] in order to turn software engineering into a true engineering discipline.

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, [10], and case studies are, for example, discussed in [13].

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 infeasible. A suitable starting point is, therefore, many times to start with studies at the universities using students as subjects, i.e. the studies 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 a study in a university setting and based on the outcome we can, as a part of a technology transfer process, replicate the study in industry and then continue with a pilot project.

2.3. Case study research within the PSP

The PSP provides some interesting opportunities for empirical studies with a well defined process, generally available descriptions, and measurements as a central theme in the process. This implies that as a natural part of the PSP, data are collected, which can be used to increase our understanding of software development. One advantage of using the PSP as a context for empirical studies is that it is well-defined, hence the design of an empirical study is more or less given through the use of the PSP [4]. The study presented in this paper is one example of how the PSP can be used as a context for case study research. It is also possible to use the environment for experimentation. Some examples of using the PSP for experimentation are presented in [7, 8].

The main objective is to improve our understanding of the defect detection process in general and within the PSP in particular. Thus, the intention is both to understand the use of the PSP and to study the effect of techniques and methods introduced through the different main increments in the PSP, for example, the introduction of reviews between PSP1 and PSP2. This is achieved through a series of minor studies related to the defect detection process. In this paper, four different aspects related to software defects and defect detection are presented. The four aspects are:

- distribution of software defect detection. Where are defects found? How does the distributions change as the PSP is improved? This is further elaborated in Section 4.1.
- defect evolution is discussed in Section 4.2. How does the number of defects change over the 10 assignments? Does the process improvement mean that the number of defects decreases?
- defects and the main PSP steps. How does the defect density change between PSP0, PSP1 and PSP2? How does the cost of software detection activities evolve? These issues are further discussed in Section 4.3.
- background of individuals versus the outcome in number of defects and defect density. How does the background of the individuals influence the number of defects? How does the background affect the defect density? Section 4.4 provides a discussion of these issues.

3. Case study

3.1. Introduction

The case 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 [4], 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 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.

Data were collected according to the PSP [4] with some minor additions. In order to achieve the objective of this study, the following data are of particular interest:

- defects detected (total, review, compilation and test),
- program size (primarily to derive defect density),
- effort (or cost) spent on defect detection (minutes spent in review, compilation and test), and
- individual background (seven variables are defined in order to try to capture the experiences and knowledge of the course participants).

The actual measures are discussed in more detail in Section 4.

3.2. 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 general hypothesis based on experiences is that the more experiences in the field the better performance. The latter is here measured in terms of the number of defects and the defect density. In particular, it is expected that more knowledge in C will mean a lower defect density. In other words, beginners in C are expected to make more mistakes.

The background experiences are measured on an ordinal scale with four classes. For example, the C experience is measured using the following four classes, similar classes are introduced for the other measures.

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

No further planning was required as the data are collected as an integral part of the PSP.

3.3. Validity evaluation

This is a difficult area. In our particular case, we have several levels of validity to consider. Internal validity is concerned with the course this year. External validity can be divided into: students taking the PSP course in forthcoming years, 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. It is also probable that similar results will be obtained in the forthcoming years the course is given.

Concerning the other threats to the external validity, it is difficult to generalise 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 generalised to other PSP courses. The study presented is based on student data, and the results may be slightly different when involving software engineers from industry. The main results will, however, probably be valid.

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 studies:

- There is, for example, no reason that people having different background experience from a particular programming language perform differently between the PSP and software development in general. The objective of the PSP is to scale down software development to the individual level, hence it is reasonable to believe that the findings can be generalised to software development in general. A similar argument can be given for any of the investigations presented here.
- The performance measures (number of defects and defect density) can be collected for other environments than the PSP, and there is no reason that we should not get a similar behaviour for another environment. Thus, we believe that the results can be generalised to other contexts.

3.4. 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, and our intentions of analysing the data. They were, however, not aware of the actual studies. 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 [4]. The empirical study is executed over 14 weeks, where

the 10 programming assignments are handed in regularly. The data are primarily collected through forms. The 10 programming assignments are mostly small statistical programs. The complexity and difficulty of the programs vary slightly. Interviews are used at the end of the course, primarily to evaluate the course and the PSP as such.

3.5. 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 been removed not because the evaluation was based on the actual figures, but because of 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.

4. Data analysis

4.1. Defect distribution

The first step in understanding software defects in general and their detection in particular is to use descriptive statistics. This means plotting the defect data. In PSP0 and PSP1, the main defect detection methods are compilation and test. From the course, data for PSP0 and PSP1 are available from six assignments (denoted 1A-6A). In PSP2, design and code reviews are introduced. It must be noted that the reviews are introduced before compilation. In other words, the code is reviewed before compilation. Data for PSP2 are available for three assignments (denoted 7A-9A). The introduction also means that new types of defects are found. The reviews are the only means of detection defects related to the coding standard, hence defects related to not following the coding standard have not been detected before.

The percentages of defects found using the different detection methods are plotted in Figure 1. In Figure 1a, the data for PSP0 and PSP1 are plotted indicating that 58% of the defects are found in compilation and 42% during testing. The relationship between defects found in compilation and testing is interesting. The factor is: (Number of defects found in compilation) / (Number of defects found in test) = 1.38. In Figure 1b, design and code reviews are added to the picture. Most defects are still found during compilation, but it is interesting to note that the relationship between defects found in compilation and test respectively has changed. The factor is now: (Number of defects found in compilation) / (Number of defects found in test) = 1.17. This implies that the reviews have found a larger proportion of defects that would have been found in compilation than in test. This is not according to the objectives, since it is obvious that the compiler is much more effective in find-

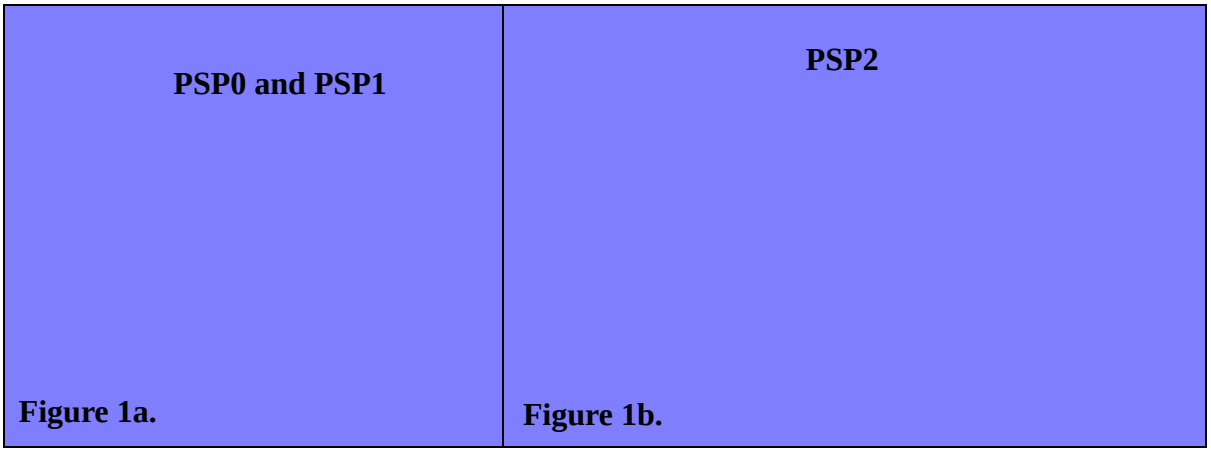

Figure 1a and 1b. Distribution of software defect detection.

ing certain types of defects than the human being, hence indicating that the reviews have to be more focused on defects that escapes the compiler. Several other studies can be found in the literature regarding the efficiency of reviews and other questions related to reviews, see, for example, [11, 12]. The former includes a comparison between different defect detection methods for requirements specifications.

To further increase the understanding of the distribution of defects found in compilation, test and in totality, the distribution of the number of students finding a certain number of defects is investigated. The distribution of the number of defects for compilation, test and in total are plotted for PSP0, PSP1 and PSP2 separately. These plots are shown in Figure 2.

From Figure 2, it can be seen that the number of students finding fewer compilation defects seems to increase; the highest bar in the diagram changes from 60 to 40 and then to 20 defects in PSP2. This is further investigated in Section 4.3, where the different main steps of the PSP are

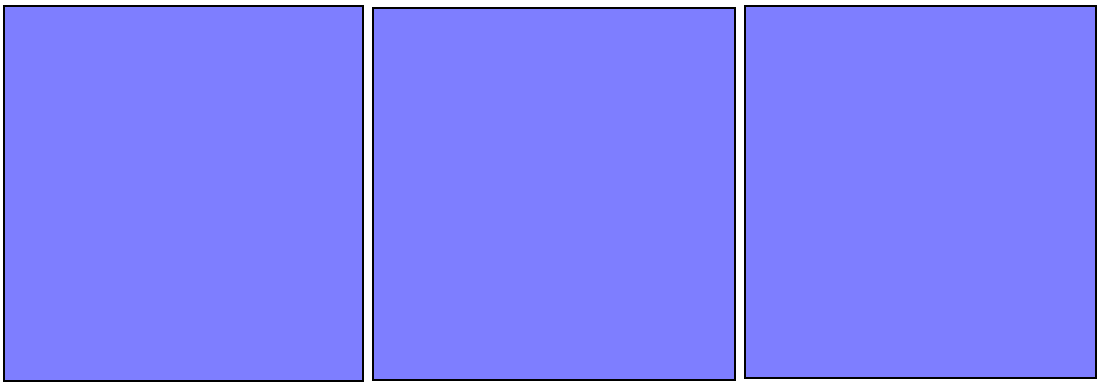

Figure 2a. Compilation, test and defects in total for PSP0.

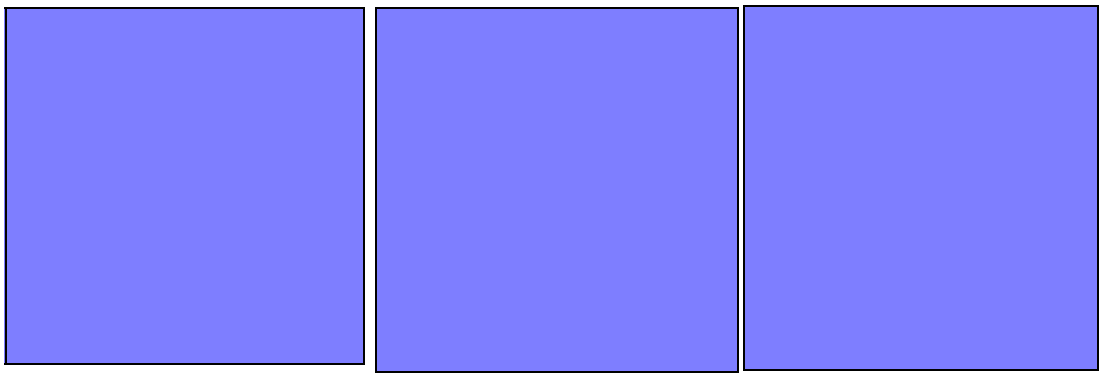

Figure 2b. Compilation, test and defects in total for PSP1.

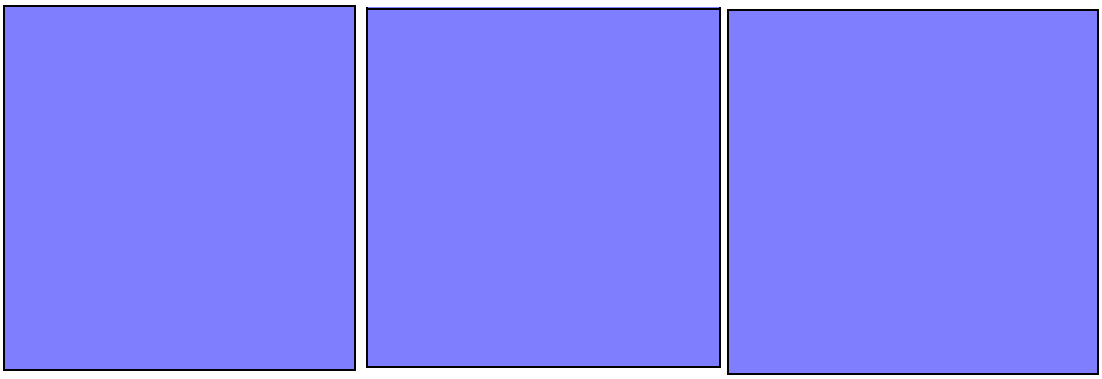

Figure 2c. Compilation, test and defects in total for PSP2.

compared in more detail. The defects found in testing seem to have a similar tendency, although not as clearly. The picture gets more blurred when looking at the total number of defects. There seems to be an improvement between PSP0 and PSP1. It is, however, more difficult when comparing PSP2 with the previous instantiations (PSP0 and PSP1). A potential explanation is that in PSP2 reviews are introduced and new types of defects are found, for example, deviations from the coding standard.

It should also be noted that in the diagrams in Figure 2, two outliers have been removed. One individual had 386 compilation defects for the three programs developed using PSP2, and the same individual has slightly more than 600 defects in total for these programs. The values are extremely high in comparison with the other students, but the data are believed to be correct. A potential explanation is that the student changed his behaviour in particular the logging of defects when entering PSP2. It should also be noted that the student was among the students having the largest problem with C throughout the course.

The above diagrams in Figure 1 and Figure 2 have provided a basic understanding of the distribution and evolution of software defect detection as the PSP evolves. In the following sections, different issues are addressed more formally and in more detail.

4.2. Defect evolution

An important question from a PSP perspective is whether the quality of the software increases as the process is improved. The question is also valid from a more general perspective, since it has to do with process improvement and its effect on quality. The main quality measure here is defect density. A major drawback with defect density as a measure is that it only focuses on defects found, and it does not take neither defects not found nor the effect in operation of the defects into account. It is, however, a reasonable measure on an individual level within the PSP course, since the programs are rather small (average size: 100 new and changed LOC), and all defects are assumed to have been found after testing and usage. It should be noted that most of the programs are reused in later assignments, hence supporting the assumption that all defects actually have been found, and that defect density is a relevant measure.

The average defect density for the 59 students and its evolution over the 10 assignments are shown in Figure 3.

In Figure 3, it can be seen how the general trend of defect density decreases through the 10 assignments, although when looking at the different main steps of the PSP, it seems like the number of defects increases with the introduction of PSP2. A potential explanation is, as stated above, the introduction of reviews and hence introduction of new defect types, and an additional explanation is the high values discussed as outliers above. This does, however, not explain the result of assignment 10 when using PSP3.

In order to model the decrease in defect density, linear regression is used. The linear model becomes:

$$DefectDensity = 105{,}9 - 4{,}687 \times ProgramNumber \quad \text{Eq. 1}$$

The model is significant ($p = 0.0046$), and R-square (quadratic correlation coefficient) is equal to 0.654. It is hence quite clear that the defect density decreases with the

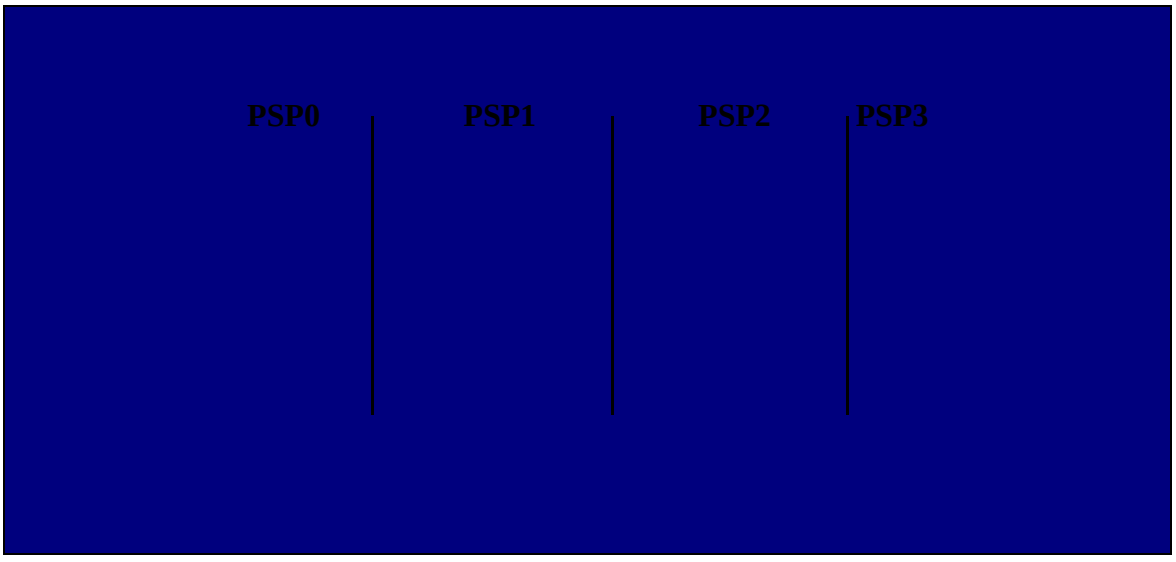

Figure 3. The trend of defect density.

assignments. From the model in Eq. 1, it is interesting to note that for each assignment the average defect density (defects/KLOC) is decreasing with 4.7 defects. The linear model is expected to become invalid if more assignments were carried out. The average defect density is expected to reach a plateau, hence breaking the decreasing trend.

A major problem is to actually explain the behaviour in Figure 3. Several explanations are possible, or combinations of them. The following three explanations are believed to have interacted:

- programming learning effect,

  This includes both the learning effect due to that several students were novices in C, and the general improvement in programming skills acquired by programming regularly. The difference between students having different experiences in C is addressed in Section 4.4. The general improvement in programming skills through the course is more difficult. The students are, however, believed to have improved their skills through the course, since most of them had not programmed recently. The PSP course is read in their fourth year at the university, and the programming courses are primarily taught in the first and second year.
- PSP learning effects.

  This effect includes that the students become more familiar with process descriptions, templates and procedures for the PSP in general. Initially in the PSP, it is several concepts to understand and moreover the students must understand the type of assignments they are given. It is difficult to judge how much this learning effect affects the defect density. It is probably the least contributing effect, but it can not be dismissed.
- PSP improvement effects.

  This is the effect to which it would be beneficial to attribute the improvement in defect density, since it would imply that the PSP as such improves software development. It is unfortunately not possible to distinguish this effect from the effects above.

Thus, it is possible to show that the defect density decreases, and that the decrease is significant. It is, however, not possible to distinguish different effects from each other, at least not in this particular case.

To further understand the improvement, it is possible to test different hypotheses regarding improvement between the main steps in the PSP. This is further discussed in the subsequent section.

4.3. Change PSP0-PSP1-PSP2

In order to evaluate the differences between PSP0, PSP1 and PSP2 more formally, hypothesis testing is applied. A statistical test is applied to show the differences with a statistical significance. The general hypothesis is that there is no difference between PSP0, PSP1 and PSP2. More formally:

- H0: Measure_0 = Measure_1 = Measure_2

where Measure denotes any measure to be compared between the different main PSP steps. The alternative hypothesis is:

- HA: Measure_0 ¦ Measure_1 or Measure_0 ¦ Measure_2 or Measure_1 ¦ Measure_2.

The objective is to be able to reject H0 and show any part of the alternative hypothesis with a statistical significance.

Five “goodness” measures were defined to investigate the differences between PSP0, PSP1 and PSP2. They are:

- Defect density in compilation
- Defect density in testing
- Defect density in total

  For PSP0 and PSP1, the total number of defects is equal to the sum of defects found in compilation and testing, and for PSP2, review defects are also included.
- Quality cost / line

  The quality cost is measured in terms of effort or more specifically the minutes spent on defect detection activities (review, compilation and testing). The objective of the measure is to capture the effort needed for defect detection per LOC for PSP0, PSP1 and PSP2 respectively.
- Quality cost / defect

  The quality cost is measured as for the previous measure, but instead the cost is normalised with the number of defects.

In order to understand the dimensions captured by the five measures defined, a principal component analysis (PCA) is carried out using an orthogonal transformation solution with varimax [9]. The results of the PCA is presented in Table 1.

From Table 1, it is clear that two main aspects are measured, namely defect density and quality cost, see grey cells in Table 1.

For each of the five measures, a factorial ANOVA (ANalysis Of VAriance) is applied based on having three treatments (PSP0, PSP1 and PSP2) for each of the variables. α is set to 0.05, i.e. if the p-value for the ANOVA is less than 0.05 then a Fisher PLSD test is applied to compare the different main steps in the PSP pairwise. The outcome of the Fisher PLSD test is also considered significant on the 95% level. In Table 2, the p-value for the ANOVA, the mean difference between the measures and whether the difference is significant or not are presented for the five “goodness” measures.

Table 1. PCA for "goodness" measures.

Variable	Factor 1	Factor 2
Defect density in compilation	0.891	-0.117
Defect density in test	0.846	0.155
Defect density in total	0.976	-0.014
Quality cost / line	0.382	0.848
Quality cost /defect	-0.319	0.871

From Table 2, it is interesting to note that regarding defects found in compilation and testing there is a significant difference between PSP0 and PSP1, and between PSP1 and PSP2. Thus, there is a significant improvement in terms of defect density for compilation and testing. For the total defect density, it is only possible to distinguish PSP0 from PSP1 and PSP2. It is not possible to show any significant difference between PSP1 and PSP2. This is probably due to that new defect types are found in PSP2 than earlier. Furthermore, it is notable that the difference is negative indicating that the defect density is higher for PSP2 than for PSP1.

It is not possible to show any statistically significant results based on ANOVA for the two measures related to quality cost, although the difference between PSP0 and PSP1 is significant for Quality cost / line using the Fisher PLSD test, see Table 2. Although the ANOVA test is not significant, it is worth noting that the quality cost for PSP0 is higher per line of code than for the others. This seems logical based on the discussions regarding learning effects above, see Section 4.2. The difference between PSP1 and PSP2 is negligible.

For the quality cost per defect, it is possible to see that the cost is highest for PSP2 (indicated by a negative difference), while the difference between PSP0 and PSP1 is small. The result is discouraging, although the differences are not significant, since improvements are expected.

The results indicate that the introduction of reviews has not been cost-effective. The main reasons are probably that new types of defects were found in PSP2, and perhaps most importantly that the students during the review mainly find defects which would have been detected by the compiler. This observation is consistent with the result in Section 4.1. It should be kept in mind that the result comes partly from the requirement to perform the reviews prior to compilation. Another conclusion from the observation is that it is necessary to emphasise on focusing the reviews on defect types which otherwise would not be found until the test phase. It is important to note that we have several explanations to the results. Thus, we should not be tempted to conclude that reviews are not cost-effective. They must, however, be better targeted at the "real" problems.

4.4. Experiences versus outcome

A basic assumption in statistical analysis is that the sample being analysed is representative of the population under study. The participants in the study are students, but they still have different background and experience, for example, knowledge in C. In order to understand the influence of previous experiences, a survey of the background

Table 2. Significant differences between PSP0, PSP1 and PSP2 respectively.

Variable and p-value for ANOVA	Pairwise	Mean difference	Significant?
Defect density in compilation	PSP0-PSP1	17.288	Yes
p = 0.0001 (significant)	PSP1-PSP2	13.978	Yes
Defect density in test	PSP0-PSP1	8.522	Yes
p = 0.0005 (significant)	PSP1-PSP2	8.691	Yes
Defect density in total	PSP0-PSP1	25.851	Yes
p = 0.0308 (significant)	PSP0-PSP2	20.607	Yes
	PSP1-PSP2	-5.205	No
Quality cost / line	PSP0-PSP1	0.388	Yes, but ANOVA is not significant
p = 0.067 (not significant)	PSP0-PSP2	0.309	No
	PSP1-PSP2	-0.069	No
Quality cost / defect	PSP0-PSP1	-0.6	No
p=0.2609 (not significant)	PSP0-PSP2	-4.492	No
	PSP1-PSP2	-3.892	No

was filled out by all students at the first lecture. Seven measures were defined. Five of them are measured on an ordinal scale with grades 1-4, one measure is on a nominal scale (two alternatives), and the seventh measure is also on an ordinal scale in terms of number of courses taken in the software area. The seven measures are:

- Study programme (scale CSE or EE)

  The students come from both the Computer Science and Engineering programme, and the Electrical Engineering programme.
- General software engineering knowledge (grade 1-4)
- Courses in the software area (outcome in the range 2-10)
- General programming skill (grade 1-4)
- Knowledge in C (grade 1-4)
- Knowledge in C++ (grade 1-4)
- Knowledge of PSP prior to the course (grade 1-4)

The objective is to study: the effect on the number of defects and defect density based on experience.

Prior to the statistical analysis regarding statistically significant relations, a PCA is carried out to understand the dimensions captured with the survey. The result from the PCA is presented in Table 3.

The measures fall nicely into two main factors, see grey cells in Table 3. The first factor captures general software knowledge and the second factor relates to programming skills. The PSP measure is not possible to place in any of the factors, and basically it depends on that most students had no knowledge of the PSP, and hence the grades were very skewed towards a low grade independent of the other measures.

The objective is to compare the experience with the outcome in terms of number of defects and the defect density. A t-test was applied for the study programme, and an ANOVA test was used for the other six measures. None of the tests were significant, hence indicating that the previous experience and background had no statistical significance on the defects. Moreover, it is particularly interesting to note that the previous knowledge in C did not significantly affect the number of defects or the defect density. This is rather surprising, but it may indicate that the PSP is rather robust and provides good support independently of the previous experience.

Table 3. PCA for the experience of the course participants.

Variable	Factor 1	Factor 2
Programme	0.823	0.343
Software engineering	0.864	0.206
Courses	0.829	0.381
Programming	0.499	0.663
C	0.062	0.890
C++	0.101	0.921
PSP	0.477	-0.158

The relationships between background and experience, and seven different performance measures are further discussed in [14]. For some of the other performance measures, for example productivity, there is a significant relationship with the previous experience.

4.5. Summary of findings

The objective of the empirical study of the PSP was to increase our understanding of the software defect detection process. The intention was, by studying different aspects of the process, to gain a greater understanding of both the PSP as such and defect detection in general. As outlined in Section 2.3, four different aspects have been studied. The main results of the studies are summarized in Table 4.

A general discussion of the findings and the gained understanding is provided in the subsequent section.

5. Discussion

From the study, it is possible to conclude that improvements in defects are obtained as the PSP is refined. It is, however, not possible to actually determine the actual cause of the improvements. The quality cost does not improve. This is seen both when normalising with LOC and defects. The introduction of reviews means that several defects that would have been found by the compiler

Table 4. Summary of findings.

Study	Findings
Section 4.1: Defect distribution (Descriptive statistics)	The mean value and standard deviations of the defect distribution decreases with the PSP level.
Section 4.2 Defect evolution (Statistical association)	The defect density decreases with the PSP levels, and a significant linear relation was found.
Section 4.3: Change PSP0-PSP1-PSP2 (Statistical inference)	A significance test shows that there is a significant decrease in defect density, but there is no significant decrease in quality costs.
Section 4.4: Experiences versus outcome (Statistical inference)	The individual experience had no significant effect on the defect density.

now are found using reviews. This is not cost-effective, hence it is necessary to focus the reviews in the PSP better on defects normally found in testing.

It is also notable that it is not possible to show any statistical significant relationship between background and experience, and the number of defects and defect density. In a more extensive study [14], it is shown that the experience does affect other performance measures.

The study has shown several interesting aspects regarding the PSP. This includes the improvements observed, the result in quality cost when reviews are introduced, and the inability to show a relationship between experience and defects. The obvious question based on the results is, of course, if the results are valid for software development in general. The following general observations are interesting for further studies both within the PSP and for software development in general:

- training, practice and improvement in combination seem to lower the defect density. This is true for compilation and test defects as well as with defects in total.
- code reviews prior to compilation must be targeted towards logical defects not detectable by the compiler to be worthwhile.
- the number of defects and defect density are not in isolation dependent on experience. In the enlarged study [14], it is shown that the productivity is higher for students with more experience. This seems to indicate that the more experienced students produce more in a shorter period of time, but they seem to make as many mistakes.

Based on the experience from this study and other empirical studies related to the PSP, we believe that the PSP provides interesting opportunities to enhance our understanding of software development. Thus, the intention is to continue to use the PSP as a context for empirical studies. The objective is to replicate the current study, extend it to other attributes, for example accuracy in estimation, and to evaluate how the findings from the PSP can be generalised to software development in general.

Finally, we would like to emphasise the need to first understand software development (some results are provided in this paper) and relationships between different parameters. The next step is to control software development based on the understanding. The control forms the basis for systematic improvement, including product, process and people. The latter includes education and training of the personnel. The process of understanding, control and improvement are essential to master the challenge of software development.

Acknowledgment

The authors are grateful to the students taking the class for valuable comments throughout the course, and their effort in collecting valid PSP data. This work is partly supported by the National Board for Industrial and Technical Development (NUTEK), Sweden, grant 1K1P-97-09673 (project: IMPROVE).

References

[1] 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.

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

[3] P. Ferguson, W. Humphrey, S. Khajenoori, S. Macke and A. Matvya, “Results of Applying the Personal Software Process”, IEEE Computer, Vol. 30, No. 5, pp. 24-31, 1997.

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

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

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

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

[8] 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, pp. 332-339, Kyoto, Japan.

[9] S.K. Kachigan, “Multivariate Statistical Analysis - A Conceptual Introduction” Radius Press, New York, USA, 1991.

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

[11] A.A. Porter, L.G. Votta, and V. Basili. “Comparing Detection Methods for Software Requirement Inspections: A Replicated Experiment”, IEEE Transactions on Software Engineering, Vol. 21, No. 6, pp. 563-575, June 1995.

[12] A.A. Porter, H.P. Siy, C.A. Toman and L.G. Votta. ”An Experiment to Assess the Cost-Benefits of Code Inspections in Large Scale Software Development”, IEEE Transactions on Software Engineering, Vol. 23, No. 6, pp. 329-346, June 1997.

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

[14] C. Wohlin, “An Empirical Study of Personal Experience versus Individual Programmer Performance”, Technical report, Dept. of Communication Systems, Lund University, 1998 (in preparation).