C. Wohlin, "Are Individual Differences in Software Development Performance Possible to Capture Using a Quantitative Survey?", Empirical Software Engineering: An International Journal, Vol. 9, No. 3, pp. 211-228, 2004. Selected and extended for special issue from International Symposium on Empirical Software Engineering in 2002.

Are Individual Differences in Software Development Performance Possible to Capture Using a Quantitative Survey?

Claes Wohlin
Department of Software Engineering and Computer Science,
Blekinge Institute of Technology,
Box 520, SE-372 25 Ronneby, Sweden
claes.wohlin@bth.se

Abstract

Software engineering is human intensive. Thus, it is important to understand and evaluate the value of different types of experiences, and their relation to the quality of the developed software. Many job advertisements focus on requiring knowledge of, for example, specific programming languages. This may seem sensible at first sight, but is it really possible to capture software development performance using this kind of simple measure? On the other hand, maybe it is sufficient to have general knowledge in programming and then it is enough to learn a specific language within the new job. Two key questions are 1) whether prior knowledge of a specific language actually does improve software quality and 2) whether it is possible to capture performance using simple quantitative measures? This paper presents an empirical study where the experience, for example with respect to a specific programming language, of students is assessed using a quantitative survey at the beginning of a course on the Personal Software Process (PSP), and the outcome of the course is evaluated, for example, using the number of defects and development time. Statistical tests are used to analyze the relationship between experience/background and the performance of the students in terms of software quality. The results are mostly unexpected, for example, we are unable to show any significant relation between experience in the programming language used and the number of defects detected.

1. Introduction

It is more than 35 years ago since the term “software engineering” was coined. We are still struggling with the same type of problems, for example, cost overrun and software defects. It is true that we are developing more complex systems today than for 35 years ago, but the basic problems are the same. New technologies and methodologies will not solve the problem; there is no silver bullet [1]. The fact still remains that the key asset is the people developing the software. This was made perfectly clear more than 20 years ago in the book by Boehm [2]. The most important factor is the people! Other studies have also reported in the large differences in performance between individuals, see for example [3, 4].

Before turning our attention to the people, we would like to highlight some general success factors, which in retrospect determine whether a specific software project has failed or become a success. A success factor is here defined as an output from a software project and it should not be mixed up with success drivers, which are important aspects during development that may form the basis for a successful projects. The following four factors are important success factors:

- Cost: primarily related to effort, i.e. person-hours, and productivity,
- Cycle time: time from development starts to delivery,
- Quality: this is a complex attribute where one important part is defects,
- Predictability: the ability to predict the other attributes.

The bottom-line is that to achieve quality in the software, it is necessary to understand which factors that affect software quality. To shed some light on one of the most important factors, namely people issues, including the importance of specific language knowledge, in relation to the above success factors, we have conducted an empirical study. Thus, we do not intend to question whether there are large differences between individuals; we view this as a known fact. However, we would like to study the effect of the background of individuals in terms of easy quantifiable information regarding, for example, programming knowledge and relate that to their ability of developing high quality software.

The objective of the study is to investigate the performance of the individuals versus their background with a particular emphasis on programming knowledge and experience. The study has been conducted within the context of the Personal Software Process [5, 6]. 65 students filled out a survey aiming at documenting their background as they entered the PSP course. After the course the outcome was measured. Seven measures were defined to capture the performance of the individuals. The performance is for most students improved through the course, whether it is a result of the PSP or practising software development in general is hard to know. The change in performance over the 10 assignments is not viewed as a problem, since the performance is measured in terms of overall performance in the 10 assignments. The seven measures were defined so that they should capture three of the above factors; it was not possible to define any measure capturing cycle time. The assignments were handed in on a weekly basis, hence making it difficult to actually measure cycle time.

The objective is to present the results from the study and discuss the outcome. It should be noted that the objective is not to study and evaluate the PSP. The PSP is used as a context in the empirical study. To evaluate the PSP, we should study the performance of individuals before and after the course. It is fairly obvious that we will see improvements during the course as course attendee start practising planning and better follow-up. A study of the defect data from the perspective of understanding defect detection within the PSP is presented in [7].

The paper is organized as follows. Section 2 describes the design of the empirical study. The analysis of the data is discussed in Section 3. The presentation includes both analyses of the survey data, the performance data and then an inference between the survey and the performance. Section 4 provides a summary and some discussions.

2. Design of the empirical study

2.1 Context: the Personal Software Process

The Personal Software Process (PSP) has gained lots of attention since it became publicly available [5]. 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 [8, 9, 10], presenting the outcome of the PSP, both from educational and industrial settings. The PSP is currently used in a number of universities and industry is also becoming interested in applying the PSP.

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

At the university, we run the PSP as an optional course for students in the Computer Science and Engineering program and the Electrical Engineering program. Most students take the course in their fourth year, and 40-70 students take the course. The main objective of the course is to teach the students the use of planning, measurement, estimation, quality control, post-mortem 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.

A major advantage in using the PSP as a context for empirical studies is that the description of the PSP is generally available, and hence makes replication of studies conducted easier. The use of the PSP as a context for empirical studies is further discussed in [11]. The study presented here is not a study of the PSP as such, but of the relationship between programming experience and performance. The use of the PSP as a vehicle for empirical studies implies that the design of the experiment is pre-defined to a large extent. For example, metrics and templates are given by the PSP. This also means that the data collection procedure is determined by the PSP. The objective of using the PSP as context for empirical studies is that the PSP should not affect the results as such. This may not be completely true, but it is very difficult to judge the effect of context, if any. This is true not only for the PSP context, but for any context in an empirical study.

In this particular study, the use of students is not critical since the objective is to study the outcome of the PSP for people having different background and experience. In particular, the differences related to educational background are evaluated. The subjects (students) are, however, not chosen by random. They are chosen based on availability, i.e. the students taken the course. This is often referred to, as being convenience sampling [12], and the study becomes a quasi-experiment due to the lack of randomisation of subjects.

2.2 Variables

2.2.1. Independent variables. As part of the first lecture in the course, the students were asked to fill out a survey regarding their background in terms of knowledge in software development, including programming experience, for example, knowledge in C. The survey does not claim to capture real knowledge and experience. The objective was to use easy quantifiable measures and evaluate whether these were sufficient to understand the performance of the subjects. The survey material is presented in Table 1. A third column allowed the students to fill an answer.

These seven measures become the independent variables in the study. The objective of the survey was to capture their both general background in software development and programming experience. A particular emphasis was put on C and C++, since C was enforced as a mandatory programming language independent of the previous knowledge of the students. Given that we enforced a language, we wanted to know whether this also meant that the students produced software of different quality based on their prior knowledge in C, C++ and programming in general. To enforce a specific programming language is not in accordance with the recommendation in [5]. This also meant that we provided a coding and counting standard. It should be noted that the background and experience are measured from an educational perspective rather than practical experience. Thus, the measures would probably have to be changed if replicating the study in an industrial context.

The general hypothesis based on experiences is that the more experiences in the field the better performance (higher quality). For example, the hypothesis is that the students having more experience in C make fewer mistakes, hence having fewer defects in their programs. Thus, in relation to the survey in Table 1, we assume that a higher grade means a better performance. These general hypotheses are formalized and evaluated in Section 3.

The results of the survey are presented in Table 2 in terms of frequency distributions of the number of students providing a certain answer. In Table 2, we may observe that the results are not optimal from an analysis perspective. A balanced result would have been better, i.e. an equal number of students giving, for example, a 1, 2, 3 and 4 respectively. It is with the results in Table 2 not possible to perform any sensible analysis when only a few individuals have provided a certain answer. The analysis would only reflect how these few individuals differ from the rest, and it will not say anything about that type of background in general.

Table 1. Student characterization

Area	Description
Study programme (denoted Programme)	Answer: Computer Science and Engineering or Electrical Engineering
General knowledge in computer science and software engineering (denoted SE general)	1. Little, but curious about the new course 2. Not my speciality (focus on other subjects) 3. Rather good, but not my main focus (one of a couple of areas) 4. Main focus of my studies
General knowledge in programming (denoted Programming)	1. Only 1-2 courses 2. 3 or more courses, no industrial experience 3. A few courses and some industrial experience 4. More than 3 courses and more than 1 year industrial experience
Knowledge about the PSP (denoted PSP)	1. What is it? 2. I have heard about it 3. A general understanding of what it is 4. I have read some material
Knowledge in C (denoted C)	1. No prior knowledge 2. Read a book or followed a course 3. Some industrial experience (less than 6 months) 4. Industrial experience
Knowledge in C++ (denoted C++)	1. No prior knowledge 2. Read a book or followed a course 3. Some industrial experience (less than 6 months) 4. Industrial experience
Number of courses (denoted Courses)	A list of courses was provided and the students were asked to put down a yes or no whether they had taken the course or not. Moreover, they were asked to complement the list of courses if they had read something else they thought was a particularly relevant course.

Table 2. Number of individuals giving a certain answer.

Independent variable	1	2	3	4
Study programme	CSE: 32 and EE: 27			
SE general	1	14	22	22
Programming	7	28	23	1
PSP	18	30	10	1
C	32	19	6	2
C++	37	16	4	2
Courses	2: 9; 3: 6; 4: 6; 5: 7; 6: 6; 7: 11; 8: 7; 9: 3; and 10: 4.[a]			

a. Interpretation: 2: 9 means that nine students had read two courses and so forth.

2.2.2. Dependent variables. Seven measures were defined as dependent variables, i.e. measures we would like to measure and evaluate to assess if the independent variable has a statistically significant effect on them. The measures are all derived from the data collected within the PSP. The measures are all based on all 10 assignments, i.e. we do not judge performance on an individual assignment, and instead we have chosen to study the results of all 10 assignments together. The intention is to remove some of the variations with looking at individual assignments and instead capturing the overall performance in the course. The seven measures are:

- Time - total development time (often also referred to as effort and here measured in minutes),
- Size - total size of the programs, i.e. new and changed lines of code are calculated,
- Defects - total number of defects,
- Defects/KLOC - the number of defects per 1000 lines of code,
- Productivity - total size of the programs divided by the total development time (presented as LOC per hour),
- Predictability size - the relative error estimating program size (measured in percentage),
- Predictability time - the relative error estimating development time (measured in percentage).

No further planning was required as the data are collected as an integral part of the PSP. In other words, all of the seven measures defined could be derived from the data collected during the PSP course. It was decided to stick with the original measures in the PSP instead of formulating new measures. The mean and standard deviation of the seven dependent variables are shown in Table 3.

Table 3. Summary of dependent variables.

Dependent variables	Mean value	Standard deviation
Size	984	244.1
Time	3353	1342.1
Defects	75.8	63.8
Defects/KLOC	75.9	50.5
Productivity	20.0	8.3
Predictability size	37.1	15.2
Predictability time	30.7	12.2

In addition, an example of a box plot illustrating the individual differences is provided in Figure 1. Moreover, the relation between the best performing student and the worst performing student is shown in Table 4 together with the relation between the upper quartile (i.e. 75%, denoted Q3) and the lower quartile (i.e. 25%, denoted Q1) as well as relation between the 90% (denoted P90) and 10% (denoted P10) percentiles. The objective is to show the degree of individual differences in terms of performance with respect to the seven dependent variables.

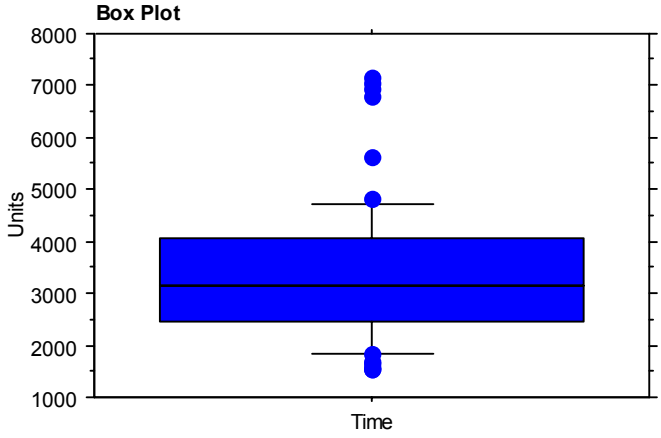

Figure 1. A box plot over the individual differences in performance with respect to time.

Table 4. Relations between individuals.

Dependent variables	Max/Min	P90/P10	Q3/Q1
Size	3.06	1.78	1.38
Time	4.62	2.57	1.65
Defects	19.15	3.46	1.81
Defects/KLOC	17.09	2.78	1.69
Productivity	5.90	3.10	2.00
Predictability size	5.49	2.79	1.83
Predictability time	5.42	2.76	1.75

Both Figure 1 and Table 4 show clearly that there are indeed large differences between the individuals in this particular data set. The very high values for Defects and Defects/KLOC are probably due to that some individuals log very few defects and others log everything. However, the other values in Table 4 clearly shows that the differences are large between, for example, the 90% percentile and the 10% percentile. It is for most measures in the range between 2.5 and 3.1. Thus, this means that between the best 10% (on a specific measure) and the worst 10% there is a difference close to a factor three. The research question is however if these differences could be explained with some simple background and experiences measures.

2.3 Hypotheses

Based on the independent and dependent variables defined we are now able to state our hypotheses better. Our general hypothesis is that more experience (higher score on the survey) means better performance, for example, fewer defects.

This means that we would like to test if there is a statistically significant relationship between the different independent variables and the dependent variables. Before doing this, it is important to evaluate which of the four success factors, see Section 1, the dependent variables represent. This is done in Section 3.2 using factor analysis after having validated the data set in Section 3.1. Prior to analysing the data, it is, however, important to address the validity of the study.

2.4 Validity

The validity of empirical studies is always important. It is, however, particularly important when the empirical studies are based on non-random samples. In software engineering, we have mostly to accept that we are unable to have random samples. This type of studies is sometimes referred to as quasi-experiments [12, 13]. In our particular case, we are interested in several aspects related to validity.

Firstly, we must consider the internal validity, i.e. are the results trustworthy? The internal validity within the course is probably not a problem, except that the measures of experience in the survey may not mirror the important aspects, although they try to capture educational aspects. The performance is measured on an individual level, and the same individuals have participated in the survey. Thus, the coupling between the treatment (difference in background) and outcome (per-

formance) is clear. Moreover, the large number of tests (equal to the number of students) ensures that the results become trustworthy.

Secondly, the external validity may be considered. The external validity is concerned with the possibility of generalizing the results outside this particular study. Two different generalizations are of particular interest 1) students entering software industry, and 2) software engineers in industry. The first generalization is the primary concern in terms of external validity in this paper.

The results from the study are probable generally valid for students entering the software industry. The students taking the PSP course at the university are probably a representative sample of students leaving the university for a software development job in industry. The Swedish students are probably representative of fourth year's students in several other countries. The software industry is very international and the Swedish students perform as good as students coming from other countries. This assessment is based on qualitative information obtained when talking to people working in the telecommunication industry. Thus, the study provides some insight of what industry can expect when hiring new employees coming directly from the university. In particular, industry obtains information regarding the value of, in terms of producing high quality software, that an applicant knows or does not know a particular programming language. For example, the study provides insight into whether it is important or not to state, in a job opening, that knowledge in a specific programming language is important.

The study presented is based on student data, and the results may be slightly different when involving software engineers from industry, but we believe that the main results are valid also for software engineers in general, since differences in background exist in all work places. In summary, we believe that the results have a rather good external validity, hence making the results generally interesting. To further investigate the validity, we would like to encourage replication of the study.

2.5 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, 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 [5]. 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. Analysis of the empirical study

3.1 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 did, hence it was judged that the extra advice might 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.

3.2 Analysis of dependent variables

The dependent variables, i.e. the seven performance measures, are analysed using a factor analysis. The objective is to evaluate the relationship between the seven measures. This is done to ensure that the seven dependent variables represent different dimensions of achieved performance in terms of different quality aspects. This is done to ensure that the measures really capture several of the many dimensions of software quality. The factor analysis is made using principal component analysis with Orthogonal Transformation Solution-Varimax, and using an eigen value > 1 as the criterion for including a factor [14]. The results are presented in Table 4. It is of particular interest to assess if we are able to capture the success factors discussed in Section 1.

Table 5. Factor analysis of the performance measures.

Dependent variable	Factor 1	Factor 2	Factor 3
Time	0.676	-0.527	-0.131
Defects	0.963	0.139	-0.024
Defects/KLOC	0.919	-0.068	0.008
Size	0.382	0.736	-0.142
Productivity	-0.301	0.925	0.037
Predictability size	-0.098	0.060	0.776
Predictability time	0.045	-0.095	0.774

The seven measures are divided into three factors. Loadings above 0.6 are shaded, and indicate which measures that are most closely related. The first factor is primarily related to defects (cf. quality in Section 1, i.e. primarily product quality). The time is included in this factor that may be viewed as unexpected, but there is a natural explanation. The long development times come primarily from the times when the students have some problems with one or two defects. Thus, the development time is very much driven by the mistakes made by the students when developing the

programs. The second factor is mostly related to size. This is not found among the factors in Section 1, but it is most closely related to cost and effort. Finally, the third factor is clearly a predictability factor (cf. Section 1).

It is interesting to note that we are able to identify two of the factors in Section 1 rather easily, and even the third is detectable. It indicates that the defined measures do indeed capture at least three success factors rather nicely. Thus, the defined measures are relevant measures of the performance of the individuals, i.e. the seven measures capture several important facets of software quality.

The correlations between variables within a factor are not as high as may be suspected. The correlation between Defects and Defects/KLOC is high, but the others are around 0.5 except for the correlation between the two predictability variables which is as low as 0.22.

3.3 Hypotheses assessment

The next step is to test the hypotheses stated above in Section 2. In the statistical inference, we use an ANOVA test to evaluate the hypotheses. These tests are parametric tests, but they are mostly rather robust, see, for example, [15]. A significance level of 0.05 is used for all tests, which is a standard level of significance in many research disciplines. The p-values obtained from the tests are shown in Table 4. If the ANOVA test turns out to be significant, a Fisher PLSD (Protected Least Significant Difference) test is performed to evaluate the pairwise significance between the measures [16]. For example, is there a significant difference between students having given a grade of 2 respectively 3 in experience in C with regard to the number of defects?

If the p-value is less than the chosen significance level (0.05), then the null hypotheses can be rejected. The study includes a large number of hypotheses given the number of variables. To illustrate the hypotheses:

- H_0: There is no difference in the number of defects based on the experience in C. Let Defects(1), Defects(2), Defects(3) and Defects(4) be the number of defects when the grade in experience is 1, 2, 3 and 4 respectively. The hypothesis can now more formally be stated as: H0: Defects(1) = Defects(2) = Defects(3) = Defects(4)
- H_A. The alternative hypothesis is that the experience in C does make a difference in the number of defects. This can also be formulated as that there is a difference between two or several of Defects(1), Defects(2), Defects(3) and Defects(4).

In Table 6, the significant results are shaded. It should, however, be noted that a significance level of 0.05 indicates also a 5% risk of getting a significant result although there is no significant result. This is particularly crucial since the large number of tests may very well mean that some findings are a statistical artefact rather than a true result. This also emphasizes the need for replication of the study.

The p-value is less than 0.05 in some other cases, but in these cases the significant difference comes from a few individuals having extreme values, see also the discussion regarding balance in the data set in Section 2 in relation to Table 2. Further, it is of course important not only to identify significant results, but also the differences (size of the effect). The latter is however out of the scope of the analysis here.

Two main issues are worth highlighting:

- Significant results,
- Unexpected non-significant results.

These two issues are discussed together. The results are discussed from the perspective of the experience data, i.e. the discussion of Table 6 is done column by column (Column 1-7).

Table 6. The p-values for independent variables when evaluated versus the dependent variables.

p-value	Programme	SE	Courses	Prog	C	C++	PSP
Time	0.098	0.034	0.394	0.173	0.101	0.318	0.022
Defects	0.602	0.335	0.260	0.701	0.768	0.961	0.928
Defects/ KLOC	0.832	0.458	0.325	0.845	0.724	0.899	0.840
Size	0.047	0.032	0.365	0.429	0.903	0.947	0.932
Prod.	0.002	0.001	0.020	0.035	0.111	0.110	0.465
Pred. size	0.015	0.138	0.243	0.243	0.166	0.183	0.507
Pred. time	0.256	0.750	0.695	0.756	0.271	0.272	0.807

Some of the aspects worth noting from Table 6 are:

1. Study programme
   Significant: three factors show a significant difference between students from the Computer Science and Engineering, and Electrical Engineering. The factors are size, productivity and predictability of size. It is interesting to note that it is the students from the Electrical Engineering programme that write the smaller programs, and they are also better in predicting the size. It is, however, the students from the Computer Science and Engineering programme that are most productive.
   Non-significant: there is no difference in the number of defects, which is unexpected. Based on the educational programme, it would be expected that the Computer Science and Engineering students make fewer mistakes due to more practice in programming. The mean value of defects is lower for the students from the Electrical Engineering programme, although the results are not significant.
2. General knowledge in computer science and software engineering
   Significant: the significant results for the development time is a result of an extreme outlier hence they are ignored. The other significant results are between grade 2 versus grade 3 and 4. It is, however, rather unexpected that the persons giving a grade of 2 write the smallest programs, but they also have a significant lower productivity. The latter is as expected. It may be that we should reconsider if a smaller program is better or not. It may be the case that the larger programs are better structured.
   Non-significant: the other results are not that interesting. The general knowledge in computer science and software engineering was not expected to be one of the variables influencing the success factors the most.
3. Number of courses
   Significant: only the productivity shows a significant difference. The difference is between those having read 2-3 courses and those having read 7-8 courses. In other words, the more general knowledge increases productivity. This is probably due to more programming practice when taking more courses within software development. The significance is closely related to

the significance discussed above.
Non-significant: for all other variables, it is not possible to show any statistically significant results.

4. General knowledge in programming
   Significant: the programming experience only significantly affects the productivity. The significant results come between those having just read a couple of courses (grade 1), and those having some more experience (grade 2 and 3). Only one individual has a grade of 4, hence it is not possible to talk about any significant results.
   Non-significant: it is rather surprising that more programming experience does not affect the other variables, for example, the number of defects and the development time. It also seems clear that although some students have more programming experience, they have not been used to make predictions.
5. Knowledge in C
   Significant: no significant results are obtained based on the experience in C.
   Non-significant: it is unexpected that experience from the mandatory language does not significantly affect any of the parameters.
6. Knowledge in C++
   The results are rather similar to the previous item, i.e. knowledge in C. It was expected that previous experience from C++ should have been important when C was used as a mandatory language.
7. Knowledge about the PSP
   The significant results come once again from one single individual and they are hence ignored. The lack of significant results is actually expected, since the evaluation is between previous experience and performance in the assignments. The previous knowledge about the PSP is not likely to influence this relationship. Thus, it underlines that this empirical study is not about the PSP, but the relation between experience and performance.

4. Discussion

4.1 Summary

Software engineering is highly dependent on the skills and knowledge of the individuals working in the field. Thus, it is important to know what type of individual differences is important to achieve high quality. This paper has addressed the situations where students have different background and knowledge in software development in general and programming in particular. The latter includes both general knowledge and knowledge of C and C++ specifically. The objective has been to evaluate two things:

- Is it possible to capture relevant knowledge and experience, using a quantitative survey, to identify high performing individuals?
- Is prior knowledge in a specific programming knowledge important when it comes to producing high quality software?

Capturing seven background and experiences measures and comparing with the actual performance of the different individuals evaluated the first question. The second question relates to whether knowledge in a specific programming language is important for the final quality of the software or if a programming language may be learnt on the job. This was evaluated by enforcing C as a mandatory programming language within a course on the Personal Software Process.

The context of the PSP has allowed us to evaluate the dependence between programming knowledge/experience and actual performance in developing software. It has also been argued that the subjects (students) are believed to be rather representative although not being professional software engineers. How representative fourth year students are in comparison with industrial software engineers is an area for further research, which is partly addressed in [17]. The study was primarily quantitative and given the results it would have been valuable to complement the quantitative results with a qualitative evaluation. This was, however, infeasible since the analysis was primarily made after the students left the class and had moved on to other courses, Master thesis work or in some cases started working in industry.

Several hypotheses have been assessed using statistical inference. It is particularly interesting to note that the experience in the programming language had no significant effect on the performance, independent of the performance measure. In general, it can be noted that the study resulted in more unexpected than expected results. It is particularly interesting to note that no significant results were found based on knowledge in a particular programming language. Thus, it is indicated that personal ability in general is probably more important than actual skills in a specific programming language. This actually means that it is not particularly important to require knowledge in a specific programming language for different job openings. Other skills are clearly more important to develop high quality software. However, this is the result with this data set, which unfortunately is fairly skewed, and hence the previous statements should be interpreted with some caution. It would be very interesting to have the study replicated and in particular if it was possible to find a more balanced data set than the one in this study.

The results seem to indicate that the major difference in performance may not easily be measurable by different experience measures. From the data, it is clear that there are large individual differences. Some students perform better than others do, although it is not possible to show it with the defined experience measures. The actual outcome will, however, be further analysed in the future, for example, it will be studied whether the same individuals perform best for all of the measures. In other words, the future work will focus on finding different patterns in the performance measures.

4.2 Implications

Although not evaluated in the study, it is our belief that the differences between individuals are larger than that between different design methods or programming languages. The differences between individuals may very well depend on psychological issues and the ingenuity of the individuals. The latter is not easily captured in simple experience measures as evaluated in this study.

The results have implications for both those that define software engineering curricula and for those advertising and employing software engineers. From a curricula point of view, the implications are that it is more important to teach the principles of software engineering than a specific notation. Having said this, it does not mean that we should not teach up-to-date methods and techniques. However, there are important differences between having a course on “object-oriented programming using Java” or a course on “Java programming”. The former course would focus on the principles and the use Java as an example, which of course also includes teaching the semantics and syntax of Java. The latter course would probably be too focused on Java and not really give the students the full understanding of object-oriented programming as such. The recommendation based on the findings from this study would be to ensure that the principles are taught, but that up-to-date methods and techniques are used in the courses. As a consequence of the findings, it is also recommended that universities should not provide a large number of programming

courses teaching different languages based on the same paradigm; it is more important to cover different programming paradigms.

The implications for advertisers and employers are that they should really carefully consider what they want from a newly employed. How important is knowledge in a specific technique or method in comparison to having a good understanding of software development? The latter is of course much more difficult to capture, but employers who manage to do that will most certainly employ people that can more easily manage the inevitable changes in techniques and methods over time. A potential risk with focusing too much on a technique or method is that when a shift occurs to something new a person employed based on knowledge in a specific language will have a harder time to change than a person who has a more in-depth understanding of the underlying principles. These are possible implications and more studies are needed.

4.3 Further work

This reasoning leads us to the conclusion that these issues have to be studied further. First, the data presented here have to be further analysed including studying variation between individuals, different measures (for example quality in terms of defects) when some of the measures are similar (for example productivity). Second, it is necessary to evaluate the importance of people in comparison with new technologies and methodologies. Third, it is necessary to take other aspects into account when studying the performance of individual software developers. Fourth, replications are needed, i.e. studies that compare the performance of individuals with different background. It is probably not until we truly understand the human aspect of software development that we can make major progress in terms of development time and software quality.

4.4 Conclusions

It can be concluded for the first research question above that it is possibly other factors that have to be captured to explain the differences in individual performance than those that can easily be captured with a quantitative survey. Moreover, it can be concluded with respect to the second research question that the actual prior knowledge of a programming language did not affect the quality of the produced programs. The latter indicates that it is probably not worth requiring knowledge in a specific programming language in job advertisements.

Acknowledgment

I am grateful to the students at Lund University taking the PSP course for having worked hard in the course and for making a great effort in trying to follow the processes as faithfully as possible. Furthermore, I would like to express my sincere thanks to my former colleagues in the Software Engineering Research Group at Lund University in Sweden for all discussion regarding empirical studies and the Personal Software Process. Finally, I would like to thank the anonymous reviewers for useful comments that have helped improving the paper.

References

[1] F. Brooks, “No Silver Bullet: Essence and Accidents of Software Engineering”, *IEEE Computer*, Vol. 20, No. 4, pp. 10-20, 1987.

[2] Boehm, B. W., *Software Engineering Economics,* Prentice-Hall, 1981.

[3] F. Brooks, “Studying Programmer Behavior Experimentally: the Problems of Proper Methodology”, *Communications of the ACM*, April 1980.

[4] B. Curtis, “Measurement and Experimentation in Software Engineering”, Proceedings of IEEE, September 1980.

[5] Humphrey, W. S., *A Discipline for Software Engineering*, Addison Wesley, 1995.

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

[7] 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.

[8] P. Ferguson, W. S. 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.

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

[10] A. Wesslén, “A Replicated Empirical Study of the Impact of the Methods in the PSP on Individual Engineers”, *Empirical Software Engineering: An International Journal*, Vol. 5, No. 2, pp. 93-123, 2000.

[11] C. Wohlin, “The Personal Software Process as a Context for Empirical Studies”, *IEEE TCSE Software Process Newsletter*, pp. 7-12, No. 12, Spring 1998.

[12] Robson, C., *Real World Research: A Resource for Social Scientists and Practitioners-Researchers*, Blackwell, 1993.

[13] Wohlin, C., P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell and A. Wesslén, *Experimentation in Software Engineering – An Introduction*, Kluwer Academic Publishers, Boston, USA, 1999.

[14] Kachigan, S. K., *Statistical Analysis: An Introduction to Univariate & Multivariate Methods*, Radius Press, New York, 1986.

[15] L. C. Briand, K. El Emam and S. Morasca, “On the Application of Measurement Theory in Software Engineering”, *Empirical Software Engineering: An International Journal*, Vol. 1, No. 1, pp. 61-88, 1996.

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

[17] M. Höst, B. Regnell and C. Wohlin, “Using Students as Subjects - A Comparative Study of Students and Professionals in Lead-Time Impact Assessment”, *Empirical Software Engineering: An International Journal*, Vol. 5, No. 3, pp. 201-214, 2000.