How to Rate Programming Skills in Programming Experiments? A Preliminary, Exploratory Study based on University Marks, Pretests, and Self-estimation

Sebastian Kleinschmager
University of Duisburg-Essen
Institute for Computer Science and
Business Information Systems
Schützenbahn 70
45117 Essen
sebastian.kleinschmager@stud.uni-due.de

Stefan Hanenberg
University of Duisburg-Essen
Institute for Computer Science and
Business Information Systems
Schützenbahn 70
45117 Essen
stefan.hanenberg@icb.uni-due.de

Abstract

Rating of subjects is an important issue for empirical studies. First, it is desirable for studies that rely on comparisons between different groups to make sure that those groups are balanced, i.e. that subjects in different groups are comparable. Second, in order to understand to what extent the results of a study are generalizable it is necessary to understand whether the used subjects can be considered as representative. Third, for a deeper understanding of an experiment's results it is desirable to understand what different kinds of subjects achieved what results. This paper addresses this topic by a preliminary, exploratory study that analyzes three different possible criteria: university marks, self-estimation, and pretests. It turns out that neither university marks nor pretests yielded better results than self-estimation.

Categories and Subject Descriptors H.1.2 [**User/Machine Systems**]: Human factors, Programming psychology

General Terms Experimentation, Human Factors.

Keywords Programming, Experimentation, Human Subjects.

1. Introduction

Experimentation [10, 11, 15, 8] in the area of software engineering with humans as subjects often suffers from the problem that it is frequently not clear how far the results of the experiment can be generalized to other subjects or whether the result of the experiment can be generalized to any other sample at all. The main problem here is that it is unclear how subjects can be rated, i.e. what criteria can be applied in order to distinguish different qualities of subjects (see for example [10]).

In the area of experiments that measure some skills of programmers such as development time for a certain piece of software, the problem seems to be even more obvious: Different subjects have different programming skills than others. As a consequence, it seems to be clear that good programmers perform better than bad programmers. For the results of an experiment it is necessary to state what kinds of subjects have been used within the experiment. Furthermore, it is desirable to determine how many good and bad programmers participated in the experiment.

Hence, the motivation for finding distinguishing characteristics for developers is mainly the following:

- **Experimental design:** Block designs with single or multiple blocking variables (see [10, 11, 15, 8]), which can be considered as a more mature experimental design, permit to analyze the results of an experiment with respect to different influencing factors such as learning effect within the experiment, etc. However, a block design that separates subjects into different groups requires stable and valid criteria that can be used to build homogeneous groups of subjects. If such criteria are missing, the expected benefit of such an experimental design is problematic, especially in circumstances where the sample size is rather small. The danger is that an invalid blocking criterion, which divides the whole sample of subjects into different groups, leads to heterogeneous groups that cannot be compared.
- **Generalization of experiment results**: It is necessary to understand to what extend the results of the experiment can be generalized to any other sample of subjects.

I.e. it is necessary to understand to what extend the subjects participated in the experiment can be considered as "serious developers".

- **Detailed analysis of experiment**: It is desirable to analyze the results of an experiment in a way that distinguishes between good, average and bad developers. If for example a new technique is being studied within an experiment, it is desirable to understand for what kind of developer the new technique has what kind of impact. It seems to be intuitive that a good and challenging technique has a different impact on good developers (who may be able to benefit from the new technique's potential) than on bad developers (which might be already overstrained by using existing techniques).

The focus of this paper is on experiments that rely on the performance of programmers where students are being used as subjects. The motivation for this restriction is the background of the authors who performed already several empirical studies in the context of programming (such as [5, 6, 21]) where students have been used as subjects.

However, the authors' motivation is also driven by the fact, that very often reviewers criticize a missing characterization of the subjects being used in experiments, and a missing criterion in order to assign subjects to different groups: Reviewers typically doubt that the use of randomization is valid. Here, it is often proposed to use university marks or pretests in order to estimate the quality of subjects. Hence, the here described work is an exploratory study that checks whether one of these criteria (or both) can be considered as a valid one.

This paper addresses the question by applying three criteria (university marks, self-estimation, pretests) retrieved from the questionnaires given to subjects that participated in two previous experiments. Then, the corresponding criterion will be compared to the performance of the subjects within the experiments in order to determine whether the criteria turned out to be capable to distinguish between different qualities of students. We used self-estimation of subjects as a third criterion: We only used it in order to estimate whether this (from our point of view rather problematic criterion) is much worse than the other criteria.

The result of the study is that neither university marks nor pretests represent good criteria to distinguish different qualities of subjects – in fact, self-estimation is at least as good as both other criteria. Consequently, this paper reports on negative results: We claim that university marks or pretests are invalid criteria and should not be used in order to classify subjects.

The paper is organized in the following way. Section 2 discusses related work. Then, we give in section 3 a short introduction into the experiments that we used for this study and explain what kind of data was being selected from the subjects. The same section also discusses the threats to validity of this study. Section 4 explains how the exploratory study is being performed. Section 5 discusses the paper and finally section 6 concludes it.

2. Related Work

There are different works we are aware of in the area of rating subjects.

First, there seems to be some common understanding based on the work in [18] that there is a high variation between programming capabilities of similar subjects. Furthermore, based on the work described in [19], it is already known that subjects from the same educational background behave different in some debugging tasks.

Moreover, there are several studies that study whether students can be used as subjects at all (see for [7, 13, 1]) in comparison to professional developers. Here, the question is, whether "being a student" can be used as a distinguishing characteristic in order to classify a subject.

In the area of programming experiments such as for example experiments about pair programming (see for example [16]), the estimation of skills turns out to be important in order to build teams. In [16] the Myers-Briggs personality test [17] is being used in order to determine different kinds of developers in order to build homogeneous groups.

The same test is being studied in [20] in order to determine the impact of personality on code reviewing capabilities..

3. Programming Experiments

This section starts with a description of two previous experiments whose data is being used within this study. It is important to note that the experiments were designed for some other purposes: It is not the case that the experiments were designed for giving insights into the question according to what criteria students can be classified. Each experiment will be described by first giving a short overview of what was being studied in the experiment. Then, we briefly discuss the characteristics of the code to be implemented and give a short description of the experiment execution.

Then, we explain what data was selected from the subjects. Afterwards, we explain the general approach of this study, i.e. we describe what data we compare.

3.1 Experiment 1: Aspect-Oriented Programming (Java vs. AspectJ)

The first study [5] was designed in order to study the development for programming tasks which are typical applications for aspect-oriented software programming (AOP [4, 9]). Thereto, subjects needed to perform some programming tasks in Java as well as in AspectJ (which is a prominent example for an aspect-oriented programming language). Based on that, the development times for each task were measured and a comparison of the development times for Java as well as AspectJ was performed. According to that, the experiment was able to detect whether there was a difference in the development times.

The tasks needed to be implemented into a small target application that was written for the experiment. As a target application, we used a small game consisting of 9 classes within 3 packages with 110 methods, 8 constructors, and 37 instance variables written in pure Java (version 1.6). Each class was specified in its own file. The game consists of a small graphical user interface with an underlying model-view-controller architecture.

3.1.1 Characteristics of the programming tasks

Altogether, nine programming tasks were given to the subjects and the programming tasks represented common crosscutting concerns such as logging, etc.

A characteristic of the programming tasks was that a lot of similar code snippets needed to be inserted in different locations of the code. For example, for the logging task, an invocation to the logger needed to be inserted into all methods within the tasks. Hence the programming task can be considered to be rather trivial from the perspective of the intellectual capabilities required by the developer. Nevertheless, the programming tasks are more than just simple copy&paste actions: developers need to pass information about static types of parameters to the Logger, parameter objects, etc. From that perspective, we consider that more experienced developers should be able finish the tasks quicker than inexperienced developers (who probably do not read program code that fast as an experienced developer and who probably are not that familiar with the situation to copy similar code snippets to different code target and to change them there).

3.1.2 Experiment Execution

The experiment was performed on a set of 20 students based on convenience sampling (cf. [15]). All students were in the 5th semester or later. The experiment was performed in multiple sessions with different students (since it was not possible find a common date for all students). The subjects were divided into two groups, one group worked on the previous tasks using Java first and then AspectJ, the other group did the tasks using AspectJ first and then Java. The students used an Eclipse installation as their IDE. Furthermore, the students were equipped with a set of test cases that permitted to check whether a programming task has been fulfilled. Within the experiment, we measured the time for the Java and the AspectJ solutions.

3.2 Experiment 2: Statically typed programming languages (Java vs. Groovy)

The second study [6] was designed in order to examine the impact of type systems on programming performance with a special focus on type casts.

According to the first experiment, the tasks needed to be implemented into a small target application that was written for the experiment. In fact, two applications were written which consisted of 6 classes. The programming tasks had to be performed using the programming language Java as well as the programming language Groovy on the applications. The first application is a soccer application, the second one a classroom application.

3.2.1 Characteristics of programming tasks

A characteristic of the second programming experiment is that the code can be considered as rather easy, functional code. In contrast to the first experiment, the code to be written has hardly any code clones. I.e. the use of copy&paste was not required. Instead, the code was mainly to do some simple computation or to perform some type casts in order to invoke a certain method in a target object.

3.2.2 Experiment execution

The experiment was run on 21 subjects, 11 of them began with the Java implementation, and 10 began with the Groovy implementation. The assignment of subjects with respect to whether they began with Java or Groovy was random.

In contrast to the first experiment, no mature IDE was given to the subjects. Instead, they had to do the implementation in an ordinary editor. Furthermore, they received a set of batch files for compiling and running the application. Based on these the development time was measured for each task.

3.3 Data selected from subjects

For this study we only make use of the Java development times for each student. The reason for it is that the Java times were measured in tasks where the subjects did not need to learn a new technique.

Additionally, the students had to fill out a questionnaire, which slightly changed for the first and the second experiment. In the first experiment, the subjects had to deliver their university marks for the courses they passed. In both experiments, students were asked to give some self-estimations about their programming skills and their skills with a number of different techniques (such as programming language, IDE, etc.).

3.4 General approach of the study

The general approach of this study is that we compare the Java development times measured within the experiment with the results of the questionnaire. The general assumption here is, that the measured Java development times give the most objective measurement for the development skills of each subject. Based on that, we perform comparisons of the questionnaire's answers – more precisely we check whether the criteria university marks and self-estimation of programming skills permit to identify the capabilities of each subject.

Then, we check whether the use of tests would have been helped to identify the development capabilities of each subject. This is done by assuming that an arbitrary task would have been used as a test task. Then we check whether the results for such a test task permits to make a statement about the general development times for each subject within the experiment.

In order to check whether a certain criterion is able to distinguish different qualities of developers, we divide the subjects into three groups based on their measured results. Then, we check whether the criteria were able to identify these groups.

3.5 Threats to validity

Due to the exploratory nature of this study, there is a large number of threats to validity. We structure this section by discussing different facets of threats to validity.

3.5.1 General assumption

The general assumption for this exploratory study is, that the techniques within the experiments that are not explicitly learned (Java in both cases) give the most objective information about the developers' performance and gives in that way the best indicator for the "quality of the subjects". In both experiments, Java is not measured for the sake of "measuring something new" but only for having information for each subject about what a certain task requires "in the ordinary case".

3.5.2 Development time as qualitative criterion

Some points could be argued against using development time as a metric for the programming capabilities of subjects. First, it could be argued that the measurement was part of the experiment itself instead of having external factors that are somehow measured and compared to each other. As a consequence, it is possible that the Java measurements do not give objective insights about a student's programming performance, but rather "his programming performance in the experimental environment" which is potentially influenced by factors such as stress. Next, it could be argued that the measurement of development times is an inadequate measurement for expressing the quality of a subject within the experiment, because the quality of a developer is not (only) how fast he is with an artificial set of tools. Instead, the quality of a subject is rather a matter of how good he applies a set of different tools during software development and how well he is able to judge what tools should be used under what circumstances. An additional argumentation against this approach is that in both experiments development time was measured for building something completely new (new pieces of code). Although it was not explicitly measured in the experiments, it can be assumed that good debugging capabilities hardly play any role here (because the complexity of the piece of software to be created was quite low). Nevertheless, it seems clear that a good and experienced developer is able to debug a piece of software much better than an inexperienced developer. However, since debugging capabilities probably did not play any role, the resulting development times might be slightly misleading, because the important factor debugging was ignored. Finally, it needs to be noted that the code to be developed was rather trivial. Consequently, extraordinary knowledge about how to design programming systems probably did not influence the resulting debugging time.

Although we consider the above arguments against this approach understandable and serious, we think that the additional capabilities that were not explicitly measured in the experiments (such as debugging capabilities, etc.) still influence the measured times. For example, although debugging capabilities were not explicitly required, it seems valid to assume that good debugging capabilities directly influence the resulting development times in a positive way, because it can be safely assumed that even in those rather easy tasks developers make errors that need to be fixed.

Furthermore, we think that other characteristics of experienced developers influence the measured development times in a positive way: We think that effects such as the organization of the work, disciplined and unique way of coding, etc. influence the development time in a positive way.

One argument that rejects development time as a characteristic that separates experienced and inexperienced developers was recently given to the authors of this paper: A discussant of a different paper of one of the authors argued that experienced developers will probably not achieve the best development times, because experienced developers test their code much more intensive than inexperienced developers (which causes additional development time). Furthermore, experienced developers not only try to pass all test cases but also format their code according to commonly accepted code guidelines and format styles. Since this action also requires additional development time, experienced developers cannot be those ones that solve a certain task in the best time[1]. Although we agree that experienced developers tend to perform such actions, we do not agree that they achieve as a consequence worse results than inexperienced developers: Taking the discussant's argument into account would mean that inexperienced developers have rather the opportunity to solve programming tasks in less time than experienced developers –

[1]Argumentation of a discussant of the PLATEAU-workshop at Onward 2009.

which contradict from our point of view the notion of "experienced developers".

3.5.3 Questionnaires for students vs. professionals

It mustn't be forgotten that only students were being used in the here presented study. Consequently, it is at least unclear whether the results can be generalized to a different sample that does not consists of students – more precisely, it seems unclear whether the results will be similar for professional software developers. The most obvious part is that it seems more than doubtful that university grades and marks are important for classifying professional software developers that already have some years of experience.

3.5.4 University marks

University marks have many other influencing factors: lectures differ and change over time, lecturers change over time, etc. Is also mustn't be forgotten that even the level of difficulty for passing a certain exam potentially differs along the time. Hence, it is possible that two similar students differ largely with respect to their marks just because they were tested in different semesters.

Furthermore, due to some freedom within studies it is even possible that students from the same university were not examined in the same courses. Furthermore, it must not be forgotten that not every course in computer science is related to programming (traditionally, only a small subset is related to programming).

In our concrete situation, all students came from the university of Duisburg-Essen. Consequently, the marks (and the average marks) can be rather seen as indicators for a student's performance for this concrete institute. It is at least questionable whether the concrete marks are comparable to other institutes.

3.5.5 Self-estimation

Besides university marks, it can also be argued that the self-estimation of developers might be different than the self-estimation of students. In fact, we are not aware of what the influencing factors for the self-estimation of students are. It is possible that it is mainly based on comparisons to other students (in case the self-estimation is based on some kind of objective perception). However, it is also possible that the self-estimation is just an indicator for self-confidence – with rather terrible consequences, because in this case a self-estimation can definitively not be used as a criterion for a programmer's performance. Also, it might be the case that a subject's self-estimation depends on some cultural background.

4. Exploratory Study

4.1 Studying the impact of university marks on the performance of subjects

First, we are interested in studying the impact of the marks of university courses on the programming performance of each subject. University marks have been used for example in [22] as a blocking criterion with the argumentation that the mark is an indicator for the development skills of a student.

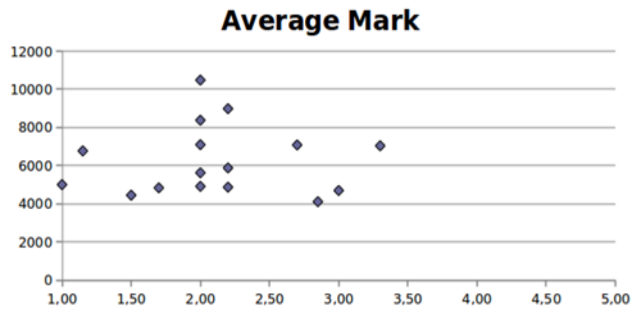

Figure 1. Development time in seconds (y) and average mark (x) of students (median)

Only the first experiment asked for concrete marks for each participant. Only 16 participants answered the questions concerning their marks, i.e. four subjects could not be considered for the following analysis.

Figure 1 illustrates for each subject its average mark (median) and its development time for all tasks in the experiment. The (ordinal) scale for marks is between 1 and 5 (according to the German education system where 1 means A according to the American education system and 5 means "failed"). Hence, students with the median of 1 are very good students while students with a median of 3 and less can be considered as rather bad students.

A first glance reveals some interesting observations. First, it looks like that students with the best marks (let's say between 1.0 and 1.7) are rather better than students with average marks – and there seems to be some logic in it and it also follows the argumentation in [22] which assumes that students with better marks are "rather better developers" as a consequence it took them less time to do the implementation tasks. However, following the same argumentation there is a contradiction: "average students" (let's say between 2.0 and 2.5) seem to take more time than students with bad marks (higher than 2.5). It is even more remarkable that a student with an average mark of 2.7 achieved the best result – while the worst result was achieved by a student with a good/moderate average mark (2.0).

Before trying to draw any conclusions from it, it is necessary to make a statistical analysis first. For obvious reasons, a (linear) correlation analysis would not give any reasonable insights (because it can be directly seen that there is no high correlation between mark and development time).

Nevertheless, it is worth to think about, whether there is a difference between the first and the second quantile (by just dividing the measurements into two quantiles). In order to test it, we performed the non-parametric Mann-Whitney-U test (cf. [3]) on the data and received a p-value of p=0.71. Consequently, there is no significant difference between both quantiles.

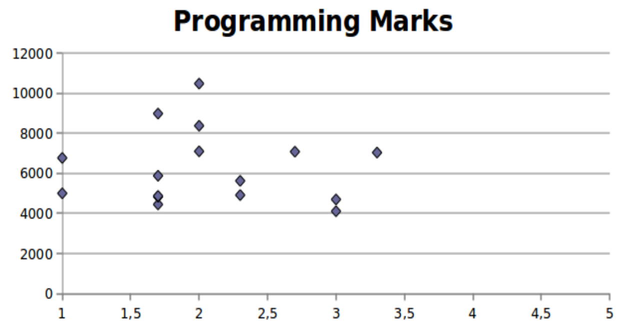

Figure 2. Development time in seconds (y) and average programming mark (x) of students (median) for programming courses

It is also desirable to check whether the previous observation that the best students (according to their marks) are better (according to the development time) than average students. In order to do so, we constructed the intervals [1–2[, [2-3[and [3-4[. Again, we performed the Mann-Whitney-U test to compare the subjects in the different groups (see Figure 2). As a result, the subjects having a mark within the interval [1–2[require significant less time (with p=0.08) than students within the interval [2–3[, students within the interval [2–3[require significant more time (p=0.10) than students within the interval [3-4[and there is no significant difference between students within the interval [1–2[and [3–4[.

Table 1. Comparison of intervals based on Mann-Whitney U-test for programming courses

Mark Interval comparison	P-value	Dominating interval
[1-2[<> [2-3[	0.14	-
[2-3[<> [3-4[	0.07	[3-4[
[1-2[<> [3-4[	0.42	-

Although we consider this result to be slightly surprising (especially that students within the last interval are better than in the middle interval) the reason for it might be, that an average mark considers all marks at the university – even those ones which do not improve programming skills. Hence, it seems to be more adequate to consider only courses which are directly related to programming.

Figure 2 gives an overview of the average marks in programming courses (median) and the development time for each subject. In comparison to Figure 1 the result is even more unclear: There does not seem to be any difference among the different groups (at least, the difference in not that obvious).

Table 2. Comparison of intervals based on Mann-Whitney U-test (dominating interval determines interval with less development time)

Mark Interval comparison	P-value	Dominating interval
[1-2[<> [2-3[	0.08	[1-2[
[2-3[<> [3-4[	0.10	[3-4[
[1-2[<> [3-4[	0.71	-

Again, dividing the result into two quantiles and performing a Mann-Whitney-U-Test does not reveal any significant result (p=0.31). The results of dividing the students in three groups according to their mark are shown in Table 2. Again, it turns out that there is a significant difference between the last interval and the middle one.

4.2 Studying the impact of self-estimation

For both experiments we asked the subjects to quantify a self-estimation. In the next sections we explain the results of comparing the self-estimations with the experimental results for each experiment.

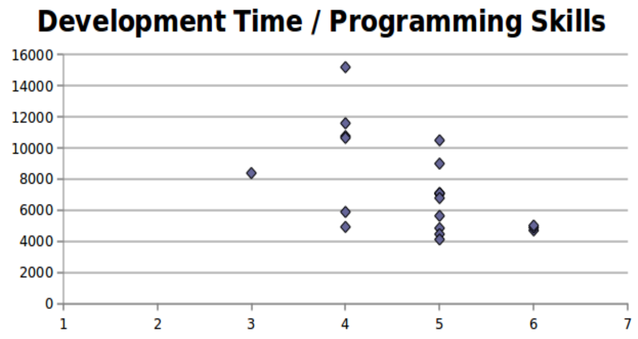

Figure 3. Development time in seconds (y) and programming skill self-estimation of subjects in the first experiment.

4.2.1 First experiment: Programming skills

In the first experiment, all 20 subjects gave a self-estimation of their programming skills on a scale from 1 (worse) to 7 (best). In the second experiment, the scale reaches from 1 (worse) to 5 (best).

Figure 3 illustrates for the first experiment the relationship between the development results and a self-estimation. It is noteworthy that none of the subjects rated themselves worse than 2 (although it must be emphasized that the students were just in the 5th semester). Three

Table 3. Comparison of programming skill and self-estimation with development time for experiment one.

Mark Interval comparison	P-value	Dominating interval
4 <> 5	0.10	5
5 <> 6	0.28	-
4 <> 6	0.05	6

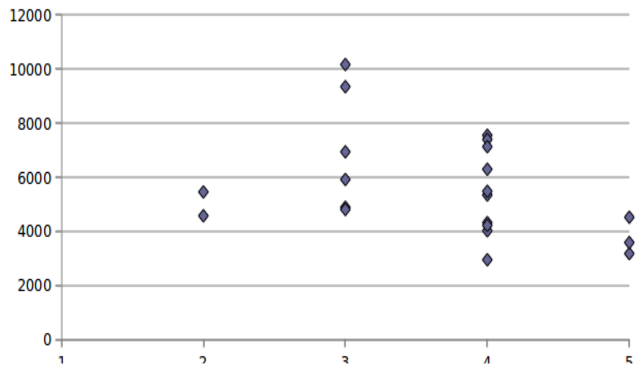

Figure 5. Development time in seconds (y) and self-estimation of programming skills in the second experiment.

subjects rated themselves as 6, only one as 3, the majority of subjects rated themselves as 4 or 5.

Although the number of subjects that rated themselves as 6 is rather low (only three of 20), it is noteworthy that it seems as if their development times are better than those ones that gave themselves less than 6. Furthermore, there seems to be the tendency that subjects that rated themselves as five require less time than those ones that rated themselves four (since only one subject rated himself as three, we do not discuss him here).

Similar to the previous section we started with a comparison of two quantiles (which reveals with p=0.6 no significant difference). Then, we compare pairwise each group with each other (ignoring the subject that rated himself as 3). Table 3 summarizes the results of applying the Mann-Whitney U-Test to the results of the experiment.

It turns out that the group of subjects that rated themselves as 5 performed better in the experiment than those ones that rated themselves as 4 and those ones that rated themselves as 6 performed better than those ones that rated themselves 4. However, no significant difference could be measured between those ones that rated themselves as 6 and those ones that rated themselves as 5.

4.2.2 First experiment: Further ratings

In addition to the programming skills we also asked subjects to rate their qualification with respect to a certain technique used within the experiment. However, the results did not reveal any special insights.

Just in order to illustrate the results of comparing other ratings with development times, we show the results of development times and a self-rating of Eclipse capabilities in Figure 4. We neglect here to give p-values for significance tests which show for all self-estimated eclipse skills no significant difference with one exception: subjects that rates themselves as 6 (a group that consists just of two subjects) were significant better than subjects that rated themselves as 3.

4.2.3 Second experiment: Programming skills

In the second experiment the 21 subjects were not asked to give their university marks. However, they were also asked to give a self-estimation about their programming skills as well as their self-estimation about their experience.

Figure 5 illustrates the results of the self-estimated development times compared to the achieved development times. It is noteworthy that the results look very similar to figure 5: at first glance it appears as if the subjects that rated themselves best (as 5) also have achieved the best development times. We have again the interesting point that a very small number of students rated themselves 2 (and which seem to dominate the students that rated themselves 3 and 4).

Table 4. Comparison of programming skill and self-estimation with development time for experiment two.

Rating Comparison	P-value	dominating rating
2 <> 3	0,2857	-
3 <> 4	0,2635	-
4 <> 5	0,1608	-
2 <> 4	0,9091	-
2 <> 5	0,08	5
3 <> 5	0,0238	5

Table 4 describes the results of the Mann-Whitney U-Test on the data by comparing each group of subjects, again. It turns out that there is only a significant difference between subjects that rated themselves 2 and 5 and those subjects that rated themselves 3 and 5.

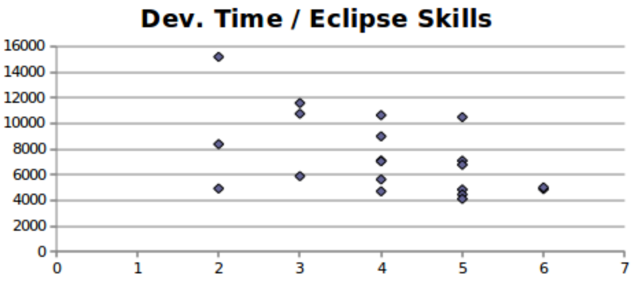

Figure 4. Development time in seconds (y) and self-estimation of Eclipse skills in the first experiment.

An analysis of the self-estimation with respect to experience reveals that there are no significant differences between different groups of subjects.

4.3 Test tasks

In both experiments, the subjects had to perform more than one programming task. It is not unusual to give subjects a test task first in order to group subjects into those ones that are relatively good developers and those ones that are relatively bad programmers. As a consequence, this implies that a single test is able to give some insights into how good a developer is.

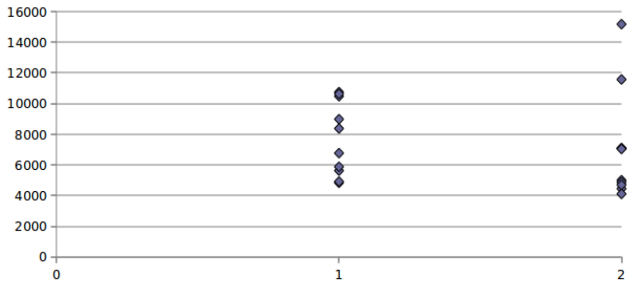

Figure 6. Subjects with best development times in in the first task (group 1) and those ones with the worst times (group 2) in comparison to the development times in other tasks.

Table 5. Mann-Whitney U-Test based comparison of significant difference of out- and underperforming subjects per task.

Task	P-value	Dominating group
Task 1	0.3	-
Task 2	0.6	-
Task 3	0.82	-
Task 4	0.45	-
Task 5	0.88	-
Task 6	0.5	-
Task 7	0.45	-
Task 8	0.71	-
Task 9	0.26	-

In this section we study, whether the use of a single test task would have been appropriate to group subjects into two quantiles.

4.3.1 First experiment

In the first experiment, nine tasks had to be done. Consequently, we are able to compare for nine times, whether the measured time for one of the tasks would have turned out to be a good prediction for the other task results.

Figure 6 illustrates the comparison of the subjects that solved the first task quickest (group 1) and those ones that solved the tasks slowest (group 2) in comparison to the sum of the other tasks. The figure already seems to indicate that there is no significant difference between both groups.

Table 5 describes the comparison of grouping the outperforming and underperforming subjects and compares them with respect to the development times they achieved in the other tasks.

It turns out, that no single task is a valid predictor for the other tasks, because for no single task the group of outperforming subjects turns out to be a group of outperforming subjects for the sum of all other programming tasks.

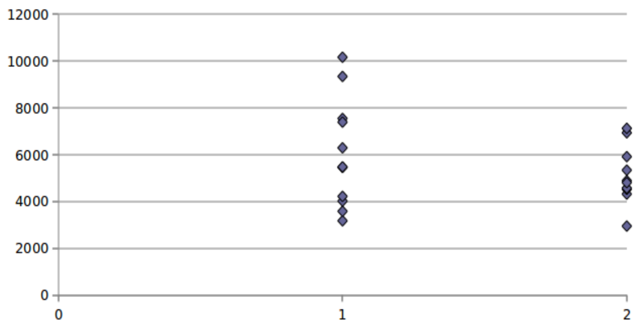

Figure 7. Development time in seconds (y) for subjects with best scores in task 1 (1) and those one with worst scores in task 1 (2) for experiment two.

4.3.2 Second experiment

In the second experiment five tasks had to be done. Again, we first give an impression of dividing the subjects into two groups according to their results in the first task in comparison to their results in the other tasks (Figure 7). It looks like there is no significant difference between both groups.

Table 6 illustrates the p-values from the Mann-Whitney U-Test which test whether for a certain task the group of best developers has better development times in the other tasks than the group of worst developers. Again, for almost all tasks there is no significant difference between the subjects with one exception – for task 2 the group of best developers turned out to achieve the best development times for the sum of other tasks.

Table 6. Mann-Whitney U-Test based comparison of significant difference of out- and underperforming subjects per task in experiment two.

Task	P-value	Dominating group
Task 1	0,4386	-
Task 2	0,0242	best dev.
Task 3	0,2599	-
Task 4	0,2313	-
Task 5	0,1213	-

5. Discussion

In this study we used two experiments which were originally designed to study some other topic and tried to evaluate how good we were able to group subjects into different groups. The study is based on the results of questionnaires that were given to the subjects in the experiment as well as on measurements of development time in the experiment. More precisely, we studied how far university marks can be used to group subjects, how far self-estimations of subjects can be used as well as how far single programming tasks can be used as tests to estimate the capabilities of a subject.

The first study (section 4.1) concentrated on the university marks of students (the information was only available for experiment 1). Here, the results were slightly surprising: students with the best marks were better than students with average marks (which seems to be understandable), but students with the worst marks were also better than the average students (which is surprising). Even more surprising is the observation that there was no significant difference between the best and the worst students with respect to the development time. It is unclear how this can be interpreted. We also analyzed the relationship between those courses that are directly related to programming. It seems obvious that these marks are more appropriate to determine the programming capabilities of a subject. However, using programming courses as distinguishing characteristic turned out to have similar problems. In fact, we were only able to detect a significant difference between students with average marks and those ones with worse marks – where students with worse marks turned out to have achieved better results in the experiment.

We conclude from that that university marks are not appropriate to determine the capabilities of subjects.

Next, we analyzed the self-estimation of subjects as a distinguishing characteristic of development capabilities. The self-estimation we concentrated most on was the self-estimated programming skill. In the first experiment, it turned out that this criterion identified at least quite good developers: Those subjects that rated themselves very high turned out to have achieved better development results than those ones that rated themselves less high.

However, likewise to the university marks, there is a set of subjects which can be considered as the average group (which rated themselves as 5). For this group there is no significant difference to the development times of those subjects that rated themselves highest.

Comparing the criteria university marks and self-estimation, it turns out that the self-estimation was able to describe the programming capabilities of subjects slightly better: It permitted to identify three groups where a pair-wise comparison reveals that the worst and the best group are significantly different – with the characteristic that those ones that rated themselves to be worse also had worse results with respect to development time.

Using the same criterion in the second experiment (whereby the scale was slightly different to the first experiment) reveals a very similar result. Again, the best groups and the worst group showed significant differences (where again the set of identified subjects was 9).

Altogether, it is possible to state that within both experiments the use of self-estimations, considering only the best and the worst groups, would have permitted to identify more than 40% of the subjects in the right way.

Finally, we studied the use of a pre-test task as a distinguishing characteristic. Thereto, we have chosen one of the tasks as a test task and separated the subjects according to their development times for these tasks into two groups. Then, we compared whether both groups differ with respect to the development times of the other tasks. It turned out, that in the first experiment, not a single task turned out to be able to define sets of developers that significantly differ. In the second experiment, only one task was able to achieve this. Hence, among 14 tasks (for both experiments) only one task was able to identify different developers.

6. Conclusion

The intention of this was to study criteria to be used in order to classify subjects in programming experiments. In order to do so, we used data collected from questionnaires as well as development time measured in previous experiments in order to determine whether one of these criteria would have been turned out to be a valid distinguishing characteristic.

The results of the study are that university marks as well as test tasks turned out to be inappropriate: It turned out that too many subjects could not be classified according to these results. The use of university marks revealed a strange result: The best students as well as the worst students turned out to be the best developers while average students achieve significant worse results.

Finally, it seems that self-estimation is not worse (and even potentially better) than the other criteria. However, we did not consider self-estimation as a valid criterion – we only used this as an alternative criterion in order to compare both other criteria to it. Hence, our interpretation of the study is that neither university marks nor pretests must be used in order to classify subjects.

Consequently, this study is a report about a negative result of an experiment. From our point of view this is not problematic – in fact it is the other way around: We are often asked to use either university marks or pretests in order to classify subjects. This study is a first indicator that both criteria are invalid.

Nevertheless, we think that it is absolutely necessary to find such classification criteria for subjects in order apply more mature experimental designs. However, we do not have a current proposal what these criteria could be – hence our proposal is to rely on pure randomized designs in experimentation as long as no such criteria are found.

Finally, it should be mentioned that this study just relies on relatively few data – just two different experiments.

Consequently, it might be too early to draw some conclusions from the data. Furthermore, the here proposed analysis is based on the comparison of quantiles. In case more data is available, it seems to be more appropriate to perform a regression analysis. In the near future we plan to apply the same analysis on two more experiments we recently performed.

References

[1] Boehm-Davis, D.; Ross, L.: Approaches to Structuring the Software Development Process, International Journal of Man-Machine Systems, 1991.

[2] Bortz, J.: Statistik für Sozialwissenschaftler, 5th ed., Springer, 1999.

[3] Conover, W. J.: Practical nonparametric statistics, 3rd edition, John Wiley & Sons, 1998.

[4] Filman, R.; Elrad, T.; Clarke S.; Aksit, M. (eds.): Aspect-Oriented Software Development, Addison-Wesley Longman, Amsterdam, 2004.

[5] Hanenberg, S.; Kleinschmager, S.; Josupeit-Walter, M.: Does Aspect-Oriented Programming Increase the Development Speed for Crosscutting Code? An Empirical Study, Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement, Florida, USA, 2009, pp. 156-167.

[6] Stuchlik, A., Hanenberg, S.: Static vs. Dynamic Type Systems: An empirical study about the relationship between type casts and development time, Dynamic Language Symposium, Portland, Oregon, 2011,

[7] Höst, M.; Regnell, B.; Wohlin, C.: Using Students as Subjects—A Comparative Study ofStudents and Professionals in Lead-Time Impact Assessment, ESEM 2000, pp. 201 - 214

[8] Juristo, N.; Moreno, A.: Basics of Software Engineering Experimentation, Kluwer Academic Publishers, 2001.

[9] Kiczales, G.; Lamping, J.; Mendhekar, A.; Maeda, C.; Lopes, C.; Loingtier, J.-M.; Irwin, J.: Aspect-Oriented Programming. Proceedings of European Conference on Object-Oriented Programming (ECOOP), 1997, pp. 220-242.

[10] Pfleeger, S. L.: Experimental Design and Analysis in Software Engineering, Software Engineering Notes, vol 20, No 3, 1995, pp. 13-15.

[11] Prechelt, L.: Kontrollierte Experimente in der Softwaretechnik, Springer, 2001.

[12] Shull, F., Singer, J., Sjøberg, D. (eds.), Guide to Advanced Empirical Software Engineering, Springer, 2008.

[13] Svahnberg, M.; Aurum, A.; Wohlin, C.: Using students as subjects - an empirical evaluation, Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM), pp. 288-290.

[14] Tichy, W.: Should Computer Scientists Experiment More? IEEE Computer 31(5), 1998, pp. 32-40.

[15] Wohlin, C., Runeson, P., Höst, M.: Experimentation in Software Engineering: An Introduction, Springer, 1999.

[16] Katira, N.; Williams, L.; Wiebe, E.; Miller, C.; Balik, S.; Gehringer, E.: On Understanding Compatibility of Student Pair Programmers", Proceedings of ACM Technical Symposium on Computer Science Education (SIGCSE), Norfolk, VA, pp. 7-11, 2004.

[17] Bishop-Clark, Wheeler, D.: The Myers-Briggs personality type and its relationship to computer programming, Journal of Research on Computing in Education, 26(3), Spring 1994, pp. 358-370.

[18] Boehm, B.W. Software Engineering Economics. Prentice-Hall advances in computing science and technology series. Prentice-Hall, Englewood Cliffs, NJ, 1981.

[19] Pressman, R.S., Chapter 19: Software testing strategies, in Software Engineering: A Practitioner's Approach. McGraw-Hill, New York, 1992, 654–658.

[20] Devito Da Cunha, A.; Greathead, D.: Does personality matter? An analysis of code-review abilities, CACM Vol. 50, No. 5, 2007, pp. 109-112.

[21] Hanenberg, S.; What is the impact of static type systems on development time, Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU) at Onward 2009.

[22] Prechelt, L., Tichy W.: A Controlled Experiment to Assess the Benefits of Procedure Argument Type Checking, IEEE Transactions on Software Engineering 24(4), 1998, S. 302-312.