Measuring the Human Factor with the Rasch Model

Dirk Wilking, David Schilli, Stefan Kowalewski

Chair for Computer Science 11, RWTH Aachen University

Abstract. This paper presents a test for measuring the C language knowledge of a software developer. The test was grounded with a web experiment comprising 151 participants. Their background ranged from pupils to professional developers. The resulting variable is based on the Rasch Model. Therefore single questions as well as the entire test could be assessed. The paper describes the experiment, the application of the Rasch Model in software engineering, and further concepts of measurement.

Key words: Human factor, Software engineering experiment, Rasch model.

1 Introduction

Having executed a few experiments (cf. [1],[2]), the authors had severe problems using variables like time, lines of code or cyclomatic complexity. One major point is that lines of code and cyclomatic complexity did not reveal a satisfying correlation with the time needed to fulfill a software task. The scale of the variables appeared problematic, too. For example cyclomatic complexity is only useful to find very difficult functions with high values. The difference between values cannot be interpreted, though. Thus, cyclomatic complexity is regarded only dichotomous in nature and lacks precision. The reliability of the variables, especially time, seems to be problematic. The reason is that while programming, experiment participants appeared to be either lucky to find a solution from scratch or to be unlucky and trying around. Thus, measurement of time has a probabilistic aspect which lowers its precision. The impression arose that a difference in the development ability between participants existed and had a more severe influence on the course of the experiment. The problem encountered when assessing participant knowledge was the imprecise variable of years of development provided by each participant. The problem with this variable is that even someone with 10 years of software development experience might not be good at it. In order to assess the influence of personal ability on software development, a solution for a measurement was sought in other disciplines in the sense of [3].

Regarding human factors in software engineering, the number of sources for this topic is scarce. One part of research based on the human factor is presented in [4], where personality types in projects were identified with a Myers Briggs Type

PREPRINT OF THE PROCEEDINGS OF CEE-SET 2007, www.cee-set.org, pp. 141 - 152, 2007.

Indicator. A further aspect of human centered research in software engineering is the cognitive aspect found for example in numerous works by Wang (cf. [5], [6]). In this area, software comprehension and reading techniques are important categories. These different approaches are represented in [7] again. In general, the human factor in software engineering is only covered lightly with several directions of interests. HCI or education related research, were the human factor is much more present, are omitted here as they do not focus on the software engineering process.

A general shift of the paradigm guiding software engineering is proposed by Cockburn [8]. A human centered approach is described while omitting any form of quantitative measurement. As this appears a step to far, a mathematical foundation for quantitatively measuring person abilities for software engineering is borrowed in the following.

2 The Dichotomous Rasch Model

In general, a test construction consists of several questions (called items from now on) measuring one variable. In this case, the variable was "C knowledge" which had to be measured using multiple questions. The answer to each question was either correct or not and this was coded as one or zero respectively. Table 1 presents an excerpt of the data. The answers of a participant are coded in one line, while each column shows the answers to the same question. The sum of the correct answers for a column can be considered a difficulty statistics of a question[1]. Simple questions are correctly answered by more participants than difficult questions. In addition to the item difficulty, the sum of a single line expresses the ability of a person. The more correct answers are given, the higher is the ability of the person concerning the measured variable. In addition, items as well as persons can be ordered based on these sums.

Participant	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8	i_9	i_{10}	i_{11}	i_{12}	i_{13}	i_{14}	i_{15}	i_{16}	i_{17}
71	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
131	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
34	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
114	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
126	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
134	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0
37	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0
70	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
84	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
20	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0

Table 1. Coding of correct and incorrect answers.

[1] Under the assumption that the Rasch Model holds.

The Rasch Model incorporates item difficulty σ_i and person ability Θ_v as parameters influencing the probability of a correct answer. Thus, the probability of a correct answer p_1 in table 1 for a person v and an item i (a specific cell) is calculated with:

$$\log \frac{p_1}{p_0} = \Theta_v - \sigma_i$$

with p_0 as probability of a wrong answer. Rearranged to p_1 it is

$$p_1 = \frac{\exp(\Theta_v - \sigma_1)}{1 + \exp(\Theta_v - \sigma_i)}$$

This model constitutes the base for further tests and algorithms ([9],[10]) as used in section 3.5.

2.1 Examples of Questions

The main challenge when creating a test for a variable is to find appropriate questions of different difficulty for the variable. As in this case questions regarding the C language were gathered, technical concepts of that language were mainly asked. These questions cover the preprocessor, pointers, call by value and call by reference, dereferencing, dynamic memory allocation, function pointers, operator precedence and so on. An example of a question is:

```
Please write down the output of this program.

int digit = 100;
int *No;

No = &digit;
printf("%d", No);
```

This question (item 10) aims at dereferencing, as in this case, the address of the variable "digit" is saved to the variable "No". Thus, the result was not a concrete value, as the address of "digit" is not given in the program fragment. Accordingly, all answers had to be open questions, in order to allow complex answers. A benefit of this step is that guessing the correct result is difficult, as simply inserting values of the program fragment often did not lead to a correct result. A drawback is that each answer had to be evaluated on its own and no automated analysis was done. Additionally, for some answers it was difficult to judge them as correct or incorrect. This was countered with strict definitions of correct answers.

Another example of a question is:

```
What kind of data structure can be stored with this definition?

char *(*(var[10]))(char *string1, char *string2)
```

Here, the participant's knowledge of function pointers is assessed (item 17). This represents the most difficult question which was supposed to need a higher degree of language knowledge. Nevertheless, it was assumed that programmers that were used to the C language directly saw the structure within the code line. Beginners and intermediate developers were assumed to not being used to functions pointers and thus giving an incorrect answer.

2.2 Advantages of the Rasch Model

One benefit of the Dichotomous Rasch Model is its simplicity together with a wide spectrum of post mortem analysis steps for a test. One example is the LR test, which compares the Rasch Model with a perfect, saturated model. If the likelihood of the Rasch Model significantly deviates from the saturated model, the Rasch Model does not hold and is rejected. Other tests focus on the homogeneity of persons and items. For example items are assumed to measure the same variable and thus are considered homogeneous in this aspect. If an easy item was too difficult for persons with higher person ability, the item might not measure the same variable and thus should be excluded.

2.3 Application to Software Engineering

Regarding software engineering programming experiments, the effect strength of novel techniques appears problematic. Novel techniques are always of major interest, but their strength sometimes is so small, that other factors mask its effect. One of the masking factors is assumed here to be the developer's programming ability. Experiences with software development projects, language knowledge, algorithm knowledge, development environment knowledge, and other person related abilities may have an effect on the time a participant needs to develop a program. Indirectly, this is shown by performance estimation of developers as done in [11]. A factor of three is reported as difference in performance with natural outliers to be found sometimes. Regarding this from a software engineering view, a length of a development task might be depending on the person executing the programming task. Finding a technique with an influence of approximately the same strength as a factor of three appears at least problematic.

The use of the Rasch Model within practical software engineering is limited. First of all, a developer's knowledge is subject to change during a project. A static assessment using a single test in the beginning thus is not appropriate. In addition, software engineering practices, projects structures, roles etc. can be changed swiftly in contrast to the abilities of a developer. Thus, person related variables are not actively changeable and out of scope.

For academic purposes, person based measuring might allow a prediction of development effort. A scatter plot of language knowledge with lines of code or program memory usage might reveal a prediction for software development. In addition, execution of a test consisting of a few questions is more economic than a pretest consisting of a complete development task. Lastly, it must be pointed out that person abilities are regarded an additional measurement variable to "hard"

variables like program reliability, project progress and time. It is regarded as an additional control device for the internal validity of studies.

3 Experimental Evaluation for the Variable C Language Knowledge

3.1 Overview

Before the actual evaluation was done, a pretest of 40 questions was done with members of the chair. Here, redundant questions (in terms of difficulty) were identified and removed. In addition, the test was subjectively regarded too difficult and easier questions were preferred. Finally, 17 questions were chosen for the final test.

3.2 Participants and Background

The experimental evaluation of the C variable was done as an online-experiment. The request for participation was posted in different bulletin boards. Regarding the participants, the choice of bulletin boards for a call for participation was critical. The aim was to include non-programmers, non-C programmers and C-only programmers in the test. The following (german speaking) bulletin boards were selected: mikrocontroller.net, c-plusplus.de, chip.de, computerbase.de, informatik-forum.at, and others with an additional group of local dormitories. Most bulletin boards were selected because of the community they represented. In addition to C language specific bulletin boards (mikrocontroller.net), general programming boards (c-plusplus.de, informatik-forum.at), general technical boards (chip.de), and non-technical boards (dormitories) are represented.

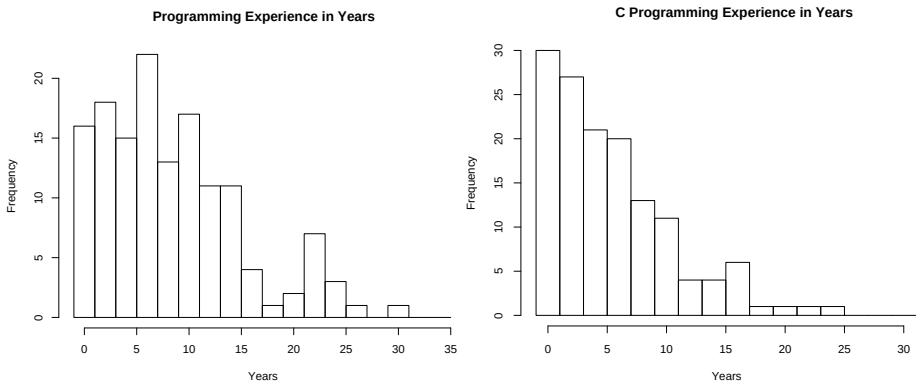

Fig. 1. Histogram of the number of years the participants were programming.

Regarding the participants' background, 151 participants are included in the study. Their general programming experience and special C language related ex-

perience is shown in Figure 1. The general knowledge of programming includes some long term programmers with a language experience of over 20 years. Regarding C, knowledge of this language is not as common compared to the general programming knowledge, which was expected.

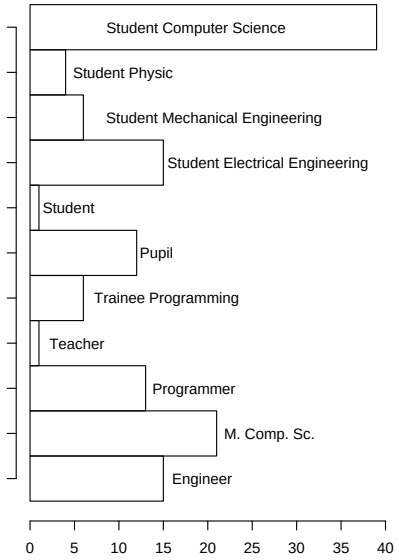

Fig. 2. Frequency of background categories for participants.

The participants' occupation is shown in Figure 2. The majority has an educational background with most in this category being students of a technical specialisation. About a third of the participants had a professional background of software development with a few engineers included in that category.

3.3 Tasks and Procedure

The task the participants had to fulfill was to answer 17 questions in an online questionnaire. In addition, some questions asked for the background of a person. This was done to check the external validity of the experiment and to check for a correlation of the measured variable and for example the years of programming. All answers were treated anonymously. At the end of the experiment, a price of €50 was given to a randomly selected participant in order to increase motivation. The average time to fill out the questionnaire was 22 minutes.

3.4 Internal and External Validity

The internal validity of the experimental study suffers from the low control possibilities during the experiment. As the test itself was only a simple website

and accordingly no additional software could be installed, participants could have used various sources like the internet sources, books, and other persons to fill out the questions. Additionally, it could not be controlled if a person filled out the questionnaire at a different computer twice.

The external validity relies on the type and quality of questions. These were based on real source code which was checked with a compiler. The kind of questions aimed at several language aspects with the language itself being standardized. The participants had a different experience level of C as described in section 3.2. In order to ground the variable, 151 participants appears as an acceptable number. One drawback is that the participants could not be selected, but their participation was based on motivation possibly leading to above average values as only "good" developers participated.

3.5 Results

The first step of analysis is the computation of the difficulty of each item. This is shown in table 2. The values are shown as logits of the probability:

$$Logit : \log \frac{p(X_{vi} = 1)}{p(X_{vi} = 0)}$$

with X_{vi} as the answer for a person v and an item i. A difficulty of zero indicates an item where the probability of correctly answering it is the same as the probability of incorrectly answering it. For the item parameter, negative values indicate easier items and positive values difficult ones. For calculating the item parameters, several algorithms and statistical programs exist. The program to calculate parameters used here is MULTIRA[2] and the results were validated using the eRm[12] package from the statistics software R with an additional self written script implementing the UCON algorithm as described in [10].

The next step consists of checking how well each item fits to the measurement model. This can be done by so called infit and outfit statistics. Outfit is a chi-square statistic which is sensitive for unexpected observations. Infit is a weighted chi-square statistic sensitive to unexpected patterns of answers (cf. [10]). Positive values indicate an underfit, while negative values shown an overfit of the according item. For the t-standardized row, values greater two are generally regarded problematic. Negative values lesser two are accepted because although they indicate a misfit to the model, an increased discriminatory power of an item is desirable. The problematic values are marked with a question mark. These items should be reworked or simply removed from the test as done below.

A graphical representation of the fitness is shown in figure 3. Here, two artificial groups are created by dividing the participants at the median person ability value. For each group, the parameter value is calculated and the corresponding values are used as coordinates in the plot. Points ideally are on a line indicating the same difficulty for both groups. While some question seem to have a good fit, other certainly need to be refined in the future to increase the quality of the test.

[2] http://www.multira.de

Item	Difficulty	Standard Error	Infit t	Outfit t
Item 1	-2.09	0.298	0.857	0.916
Item 2	-2.09	0.298	-0.413	-0.108
Item 3	-2.001	0.292	-1.152	-0.470
Item 4	-1.606	0.268	3.065!	2.277!
Item 5	-0.349	0.218	0.222	0.3
Item 6	-0.302	0.217	-2.009	-0.709
Item 7	-0.256	0.215	3.130!	3.279!
Item 8	-0.074	0.211	-0.216	-0.580
Item 9	0.057	0.208	2.373!	1.854
Item 10	0.311	0.204	-1.927	-2.311
Item 11	0.393	0.203	-2.446	-2.400
Item 12	0.555	0.201	-0.557	-1.088
Item 13	0.674	0.200	-2.127	-2.283
Item 14	0.831	0.190	0.797	0.210
Item 15	1.649	0.201	-1.530	-1.608
Item 16	1.729	0.202	-0.604	-0.645
Item 17	2.57	0.224	-0.858	-1.015

Table 2. Item parameters and fitness values.

3.6 Test Revision

In order to increase test quality, three items are removed from it. Reasons for bad items were questions which could be misinterpreted. As a correct understanding of the questions thus relied on accurate reading instead of C language knowledge, some questions did not focus on the true variable to be measured by the test. Especially for easy items, accurate reading seemed to dominate the difficulty of an item. One easy question was item four:

```
Please compute the variable solution.
In which order was the term computed?

int solution;
solution = 8 / 4 * 2;
```

In this case, operator precedence was asked for which in this case is a simple left to right execution. Although a simple question, describing the actual order was not always done correctly by persons with high ability. Thus, the question had to be removed. Item nine represents problems with correctly answering pointer based questions. Here, person ability did not seem to protect against incorrect code interpretation:

```
Please write down the value of v.

void funct( int *x ){
  *x = 5;
```

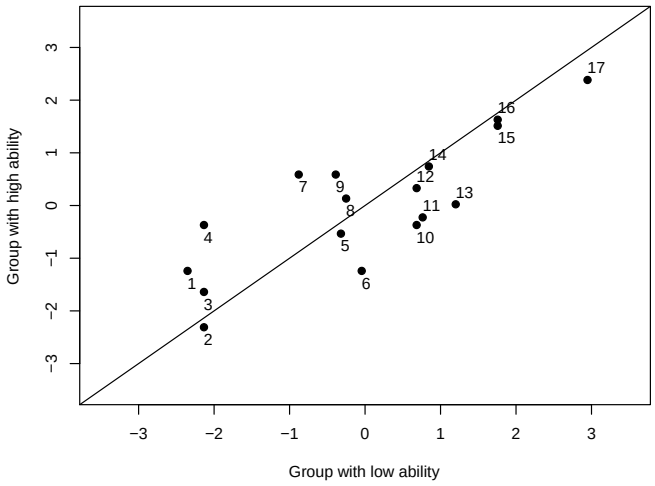

Fig. 3. Goodness of fit plot for two separated groups by median of person parameter.

```
  x = (int *)malloc(10000);
  *x = 10;
}

MainProgram:
int v = 8;
funct(&v);
print-out of v
```

In this example, the participants had to have an eye on the address a value was written to. As the variable `x` gets a new memory location, the second memory writing is not done to the global address.

By removing them, new parameter and fit values can be calculated as shown in table 3. Three questions were removed in order to get a satisfying fit to the model. The resulting difficulties show a lack of easy items between parameter values of -1 to -3 which has to be fixed in the future.

Figure 4 shows the knowledge of the C language plotted against the number of years of C programming. Here, a rough asymptotic correlation of the new variable with C experience in years can be seen. As this fits the expectation of the variable well, this gives a hint on the validity of the variable.

As the mean value of participant ability is at 0.61, the test was in general too easy. As the test does not use extreme scores, it covers 3 to 93 percent of the participants. Summing up, a c-knowledge metric was created which can be measured in about 16 minutes. The resulting variable is interval-scaled and it is grounded on 136 persons. Its value within experimentation has to be evaluated, though.

Item	Difficulty	Standard Error	Infit t	Outfit t
Item 1	-2.542	0.324	1.729	1.354
Item 2	-2.542	0.324	0.312	0.918
Item 3	-2.435	0.316	-0.657	0.092
Item 5	-0.517	0.231	1.123	1.648
Item 6	-0.463	0.230	-1.239	-0.398
Item 8	-0.206	0.223	0.636	0.85
Item 10	0.226	0.215	-0.984	-1.699
Item 11	0.318	0.214	-1.940	-2.034
Item 12	0.498	0.211	0.004	-1.156
Item 13	0.630	0.21	-1.241	-1.993
Item 14	0.805	0.209	1.810	0.678
Item 15	1.709	0.21	-1.272	-1.423
Item 16	1.797	0.211	-0.071	-0.102
Item 17	2.722	0.233	0.353	-0.496

Table 3. Item parameters and fitness values for revised test.

4 Further Concepts

C knowledge is rather easy to measure as the language C, its difficulties, syntactical problems, and important technical parts are known. Quantification of this variable thus was a conservative decision compared to other variables which are presented in the following.

4.1 Viscosity

The term viscosity is taken from [13] and [14]. It describes the resistance of a programmer to local changes. Measuring this attitude, although vague in nature, could be possible in a controlled, variable oriented way. The attitude could be assessed using situation oriented surveys where the programmer is required to make a decision which might conflict with his resistance to change. Another way to understand viscosity is the ability to find new and different solutions for problems in order to solve them. Testing this ability becomes difficult as questions for this aspect should allow open answers to gather all possible solutions of a problem.

4.2 Experience

One of the most powerful arguments of doing software engineering and omitting its pitfalls is having sufficient development experience. Measuring this aspect appears extremely difficult and methods to achieve this are mostly part of other disciplines. Nevertheless every proposed factor of influence on a software engineering project should be quantified and tested for effectiveness.

Regarding an assessment as a variable, some general thoughts are given here. A test for experience might have the general form of a situational survey based on

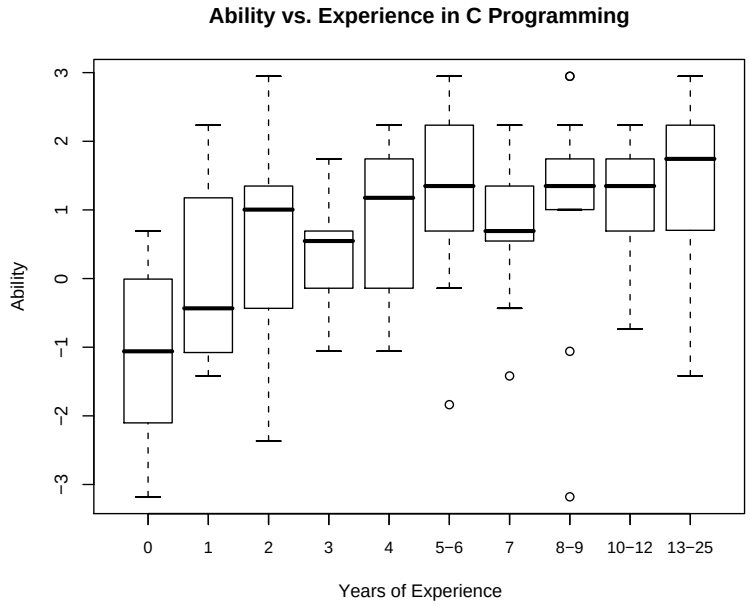

Fig. 4. Boxplots for parameter estimates of C knowledge versus years of programming.

decisions that must be made or a risk assessment that must be given. Questions must be prepared in conjunction with software engineering experts in order have a correct base for them. An interdisciplinary approach and continuous adaption of questions to incorporate technical change seem to be appropriate for this difficult variable.

5 Conclusions

This paper presents the Rasch Model as a way to assess person ability in a quantitative way. As a first step, C language knowledge was measured as a variable using multiple questions. An experiment was executed using an online survey attracting 151 participants. Using the rich evaluation possibilities of the Rasch Model, problematic questions could be identified and removed from the test.

In addition, further concepts for measurement are discussed. These comprise abstract ideas like programming viscosity and project experience. Although difficulties are expected when creating tests to assess these variables, an above average effect strength on software projects is expected from these variables.

The questions needed for the test as well as the data can be obtained from the authors.

References

1. Salewski, F., Wilking, D., Kowalewski, S.: The effect of diverse hardware platforms on n-version programming in embedded systems - an empirical evaluation. Proceedings of the 3rd International Workshop on Dependable Embedded Sytems 105/2006, Vienna University of Technology (2006)
2. Wilking, D., Khan, U.F., Kowalewski, S.: An empirical evaluation of refactoring. e-Informatica - Software Development Theory, Practice and Experimentation **1**(1) (2007) 28–44
3. Singer, J., Storey, M.A.D., Sim, S.E.: Beg, borrow, or steal (workshop session): using multidisciplinary approaches in empirical software engineering research. In: ICSE. (2000) 799–800
4. Karn, J., Cowling, T.: A follow up study of the effect of personality on the performance of software engineering teams. In: ISESE '06: Proceedings of the 2006 ACM/IEEE international symposium on International symposium on empirical software engineering, New York, NY, USA, ACM Press (2006) 232–241
5. Wang, Y.: On cognitive properties of human factors in engineering. In: Fourth IEEE Conference on Cognitive Informatics, 2005. (ICCI 2005). (2005)
6. Wang, Y.: On the cognitive informatics foundations of software engineering. In: Proceedings of the Third IEEE International Cognitive Informatics, 2004. (2004)
7. John, M., Maurer, F., Tessem, B.: Human and social factors of software engineering: workshop summary. SIGSOFT Softw. Eng. Notes **30**(4) (2005) 1–6
8. Cockburn, A.: The end of software engineering and the start of economic-cooperative gaming. Computer Science and Information Systems **1**(1) (2004) 1–32
9. Fischer, G.H., Molenaar, I.W., eds.: Rasch Models. Springer (1995)
10. Wright, B.D., Masters, G.N.: Rating Scale Analysis. Mesa Press (1982)
11. Prechelt, L.: The 28:1 grant/sackman legend is misleading, or: How large is interpersonal variation really? Internal Report 18, Universitt Karlsruhe, Fakultt fr Informatik (1999)
12. Hatzinger, R., Mair, P.: Extended rasch modeling: The erm package for the application of irt models in r. Journal of Statistical Software **20**(9) (2007) 1–20
13. Hoc, J.M., Green, T.R.G., Samurcay, R.: Psychology of Programming. Academic Press Inc. (1990)
14. Rosson, M.B.: Human factors in programming and software development. ACM Comput. Surv. **28** (1996) 193–195