Institute of Software Technology

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Individual Characteristics of Successful Coding Challengers

Marvin Wyrich

Course of Study:	Softwaretechnik
Examiner:	Prof. Dr. Stefan Wagner
Supervisor:	Dr. Daniel Graziotin
Commenced:	May 15, 2017
Completed:	November 15, 2017
CR-Classification:	K.6.1

Abstract

Assessing a software engineer's problem-solving ability to algorithmic programming tasks has been an essential part of technical interviews at some of the most successful technology companies for several years now. Despite the adoption of coding challenges among these companies, we do not know what influences the performance of different software engineers in solving such coding challenges. We conducted an exploratory study with software engineering students to find hypothesis on what individual characteristics make a good coding challenge solver. Our findings show that the better coding challengers have also better exam grades and more programming experience. Furthermore, conscientious as well as sad software engineers performed worse in our study.

Contents

1 Introduction 13
- 1.1 Motivation . . . 13
- 1.2 Research objectives and contributions . . . 14
- 1.3 Methodological approach . . . 14
- 1.4 Structure of the work . . . 15

2 Related work 17
- 2.1 Coding challenges and competitions . . . 17
- 2.2 Personality and problem-solving ability . . . 20

3 Coding challenges 23
- 3.1 Areas of application . . . 23
- 3.2 Good coding challenges . . . 24
- 3.3 Required knowledge for solving . . . 25
- 3.4 Judging solutions . . . 25

4 Methodology 27
- 4.1 Research design . . . 27
- 4.2 Participants . . . 28
- 4.3 Coding challenges . . . 29
- 4.4 Candidates for predictor variables . . . 36
- 4.5 Analysis procedure . . . 38

5 Results 41

6 Discussion 51

7 Conclusion 59

A Script for introducing participants into the study 61

B Questionnaire items 63

Bibliography 69

List of Figures

4.1 Overview of the measured variables 38

5.1 Relationship between performance and sadness 45
5.2 Relationship between performance and conscientiousness 46
5.3 Relationship between performance and B.Sc. GPA 47
5.4 Relationship between performance and DSA course grades 48
5.5 Relationship between performance and current GPA 48
5.6 Relationship between performance and programming experience 50

List of Tables

5.1 Scoring scheme 42
5.2 Performance scores 43
5.3 Correlation results for SPANE 44
5.4 Correlation results for personality 46
5.5 Correlation results for academic performance 46
5.6 Correlation results for experience 49

Listings

4.1 Coding challenge 1 30
4.2 A brute-force solution to coding challenge 1 in $\mathcal{O}(n^2)$ 31
4.3 A solution to coding challenge 1 in $\mathcal{O}(n)$ 31
4.4 Coding challenge 2 32
4.5 A solution to coding challenge 2 in $\mathcal{O}(n)$ 33
4.6 Coding challenge 3 34
4.7 A recursive solution to coding challenge 3 in $\mathcal{O}(3^n)$ 35
4.8 An iterative solution to coding challenge 3 in $\mathcal{O}(n)$ 35

1 Introduction

1.1 Motivation

Some of the biggest and most successful technology companies, including the *big four* Amazon, Facebook, Google and Microsoft, let applicants do algorithmic programming tasks as part of their technical interview process [McD15]. This might reveal how good a programmer is in finding efficient and scalable algorithms to unknown problems and show his or her ability to debug and test a small piece of source code. In the following we refer to these tasks as *coding challenges*. Although in technical interviews aspects like, for example, interpersonal skills play an important role [FBRP17], coding challenges form an essential part of the interviews and the subsequent evaluation of the candidates.

Many books, online guides, practicing websites and experience reports exist to help preparing for technical interviews and coding challenges in particular (e.g. [Dum17; Ent17; Lee17; McD15; MK+12]). To the best of our knowledge there is neither yet research on the impact of practicing coding challenges nor on which individual characteristics could influence the performance in solving coding challenges. What is known is that some software engineers perform better than others. This difference in the performance ranges from computer science graduates who can not solve simple programming tasks like the Fizz-Buzz challenge[1] to programmers who ace almost every challenge in the finals of internationally recognized programming competitions. Based on this observation, we set out to explore what individual characteristics of successful coding challenge solvers could be. Findings of our work can help students to improve early in their career in becoming better coding challenge solvers if they knew which individual characteristics have an impact on their performance. If, for example, the number of pull requests on GitHub strongly correlates with the performance in solving coding challenges, it could be the case that students should early be encouraged to contribute to open source projects.

[1] The Fizz-Buzz challenge is a trivial programming task that is used in interviews to filter out programmers with insufficient programming skills. The task is to *write a program that prints the numbers from 1 to 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz"* [Gho07].

1.2 Research objectives and contributions

The objective of our research is to identify and explore correlations between individual characteristics, such as personality or programming experience, and the performance in solving coding challenges. The contributions of this work are:

1. Persons with good exam grades are also good at solving coding challenges. We found significant moderate and strong relationships between variables related to academic performance and the coding challenge performance.
2. Sad software engineers perform worse in coding challenges. There was a significant moderate negative relationship between the affective state *sad* and the performance.
3. We observed a significant moderate positive relationship between the programming experience of a software engineer and the performance in solving coding challenges.
4. Conscientious persons perform worse in coding challenges. We found a significant moderate negative relationship between the Big Five personality trait *conscientiousness* and the coding challenge performance.

1.3 Methodological approach

In order to investigate the correlations between individual characteristics of software engineers and their performance in solving coding challenges we conducted an exploratory study with the following research question:

RQ1 What individual characteristics make a good coding challenge solver?

A total of 32 software engineering students from the University of Stuttgart took part in our study. Participants had to solve three coding challenges on a computer and between the coding challenges they had to fill out questionnaires on their current mood, personality, academic performance and prior training and experience.

Afterwards, we calculated for each participant a score for the performance in solving the coding challenges. Solutions were evaluated relatively to solutions by other candidates and according to correctness and time complexity. We then calculated the correlation coefficient between the performance score and the quantifiable data we received from the answers to the questionnaire items.

1.4 Structure of the work

The following chapter provides a summary of related work on general aspects of coding challenges, using them for educational purposes as well as in programming competitions, and on the relationship between the personality of a software engineer and his or her problem-solving ability. Chapter 3 introduces coding challenges according to our definition. We discuss what makes a good coding challenge, what they are used for, what knowledge is required for solving such challenges and how solutions usually get evaluated. In chapter 4 we describe in more detail the research methodology and the design of our study. The findings of the study are presented in chapter 5, followed by a discussion in chapter 6.

2 Related work

Related work can be divided into two broad categories. The first category includes literature on general aspects of coding challenges and programming competitions as well as work on using coding challenges for educational purposes such as automated grading of assignments and teaching algorithms. The second category comprises specific aspects of behavioral software engineering, namely the personality of a software engineer and his or her problem-solving ability.

2.1 Coding challenges and competitions

Scientific research on coding challenges focuses mainly on using them for educational purposes, such as attracting and teaching students as well as judging assignments automatically. Programming competitions are related to our work as most of them use tasks similar to the kind of coding challenges we use for our study. Work on programming competitions addresses, for example, the challenge and competition design as well as preparation and training activities. In this section we primarily focus on findings from computer science competitions in which contestants have to write code, but we also include transferable findings from different, e.g. paper-and-pencil competitions where contestants at least have to design an algorithm. Some of the most famous programming competitions are the ACM International Collegiate Programming Contest (ICPC)[2], the International Olympiad in Informatics (IOI)[3] and the Google Code Jam[4].

Researchers agree that special care should be taken when designing coding challenges and competitions [BH08; DF08; VCM+06]. The criteria vary depending on the goals and target group of the specific coding challenge or competition. Among other things, criteria for good tasks deal with the time to solve a challenge, easily understandable problem statements, the matter that the required algorithm for solving should not be too similar to classic algorithms of known problems, and that several solutions to a problem

[2] https://icpc.baylor.edu/
[3] http://www.ioinformatics.org/
[4] https://code.google.com/codejam/

of different difficulty and efficiency should be possible. Usually, strategically recalling a correct algorithm is harder than implementing it. As an example, the ICPC consists of eight to twelve challenges. The simpler ones require minimal knowledge on basic data structures, problems of medium difficulty require knowledge on algorithms, such as dynamic programming and greedy and graph algorithms, and the hardest problems combine multiple concepts and sometimes require mathematical knowledge [BS16].

2.1.1 Preparation and training

Independent of how the tasks are designed, Forivsek [For10] found that the coding challenges in competitions became harder over time. Also a pairwise comparison of almost identical challenges used in contests from different years showed that the percentage of participants which were able to solve the challenges increased and, furthermore, the average time to solve the challenges decreased [For10]. In other words, the performance level of the contestants has increased over time.
One reason could be that there now are several online resources where programmers can practice coding challenges. And there are also many books on the preparation for programming interviews that present approaches to solving coding challenges and teach relevant algorithms and data structures [ALP12; McD15; MK+12]. Additionally, some books specifically address the preparation for programming contests [Are06; SR06]. Bloomfield and Sotomayor [BS16] claim that the biggest success factor for programming competitions are training activities like working through problems and running team sessions in which problems are discussed. Unfortunately, they do not provide evidence other than by reporting their own experience. Revilla et al. [RML08] conclude, from statistics of an online judging system, that solving more coding challenges increases the individual acceptance rate and decreases the rate for wrong answers as well as for compilation errors, while the rate for submitted solutions that exceed a given time limit doesn't change. Additionally, for problems with a low acceptance rate the wrong answer rate almost stays the same - independently from the number of problems a user solved. Alexander and Izu [AI10] designed a course that teaches algorithmic techniques and uses coding challenges for practicing as well as for the exam tasks. They found that the number of practice problems solved by an individual correlates with the number of exam problems solved ($r = 0.34$, $p < 0.01$).

2.1.2 Challenges for attracting and teaching students

Using coding challenges for educational purposes and sometimes with a competitive character proved to be useful to attract and motivate student programmers [Dag10;

MN86]. The concept of algorithm efficiency with respect to running-time can be taught early and intuitively by using problems similar to those we describe in our work and described in [Gin96]. Others propose puzzle-like games for analyzing algorithms without the need to program them in a specific programming language. Students could then focus on the design of the algorithm instead of any programming syntax [CH17; Lev05]. The developed solutions, with or without source code, create a basis for subsequent discussions between the students, who generally enjoy coding challenges and seem to enjoy discussing the problems and their solutions [CJW11]. All in all, this leads to a better understanding of efficiency and offers additional insights into algorithmic problem-solving as well [Ast04; Gin96].
Paxton [Pax07] tried to improve students' problem-solving abilities with a course that used coding challenges from the ICPC for homework assignments and for the exam. In an optional survey students responded that they enjoyed this approach and that their problem-solving skills did improve during the course at least a little. Other researchers also found that assignments based on programming interview questions motivate students and by doing these kind of programming tasks they perform better in exam questions on algorithms and data structures [GF09; Urn17].
Choosing *interesting* problems is important in this educational context as it will motivate students to learn programming and participate in competitions [DS04]. Bloomfield and Sotomayor [BS16] found that most students are not even motivated by the prizes when participating in the ICPC. They saw that participating in programming contests requires skills which are valuable when applying for positions where technical interview questions are asked. And although solutions most often are only graded on their correctness but not on code quality, Astrachan [Ast04] had the impression that giving students these kind of algorithmic problems as programming assignments made them work better in larger programming projects later, and better programmers in general.

2.1.3 Automatic source code evaluation

In programming competitions automated source code evaluation plays an important role, especially in those with many participants. Among other authors, Leal and Silva [LS03] presented a system for the judging of submissions in a programming contest. It is not fully automated but it reports a classification of the submission which is subsequently evaluated by a human judge.
Apart from programming contests, automated source code assessment could also be used in programming courses. If done manually, grading and correcting of homework assignments can be error-prone, tedious and time consuming. For these reasons the use of automated online judges for educational purposes was proposed [CKLO03; TB13]. Coding challenges are especially suitable for this purpose, because the correctness of solutions to such problems can be evaluated automatically and for judging the run time, a

fixed time limit can be given in which a test case has to complete. By running automated test cases on a submitted student's solution the authors of [CKLO03] were able to increase the number of assignments whereby students had more practice. Additionally, by using an automated online judge students receive immediate and objective feedback on their submissions. Tonin and Bez [TB13] noticed that students were very enthusiastic about their online judge for self-study as students could rank up by solving more problems. Rosales et al. [RCC+16] also presented an interactive tool for judging code submissions made by students. Their aim was to give students an overview of the possible solutions and to group similar submissions automatically. A cluster graph visualization could then for example show if several students made the same mistake and thereby help instructors to give better feedback to the students. With the tool one can also analyze the student's attempts and how he or she approached a specific problem to finally get to a solution. Helminen et al. [HIKM12] found that although almost all students were able find a solution to some kind of programming learning puzzles there is a notable variance in the paths students take to get to this solution. Hence, the authors thought that inspecting the solving path is more meaningful than looking at the final result when it comes to assessing the programming skills.

2.2 Personality and problem-solving ability

Programming contests and their challenges can be used for assessing the problem-solving ability of a programmer and to keep track of their improvement over time [CJW11]. In the previous section we reported on evidence that practicing coding challenges improves the performance of an individual. However, why different programmers vary in their performance when solving coding challenges might also depend on other aspects, such as personality and emotional state. As an example, Graziotin et al. [GWA14] found that happier software developers perform significantly better in analytic problem solving. There is an increasing interest in research on the influence of a software engineer's personality [CSC15]. Personality characteristics and individual performance together make up a quarter of the investigated topics in this research field [CSC15]. It should be noted that *programming performance* in this related work section mostly refers to observations over a longer period than the usual short period of time a contestant would have in a programming competition or an interviewee would spend on a coding challenge in an interview.

It has been shown that in terms of productivity, the performance of the best programmers is several times better than that of the worst programmers [Gla02]. DeMarco and Lister [DL13] conducted a survey in which software developers from different organizations compete in a coding task that has to be completed in minimal time. They observed that

the best performance was 2.1 times better in terms of time than the average and about nine times better than the worst performance. Furthermore, for participants with more than six months of experience they found that there was no correlation between the years of programming experience and the performance. Those participants with less than six months' experience performed worse than the rest. Back in 1968, Sackman et al. [SEG68] already found that developers differ considerably in their performance, but neither found a correlation between the performance of experienced programmers and trainee class grades nor between the performance and the score in programming ability tests like the BPKT[5].

Evans and Simkin [ES89] found that the cognitive style according to Myers-Briggs type indicator is a statistically significant explanatory factor for mastering computer concepts. Cegielski and Hall [CH06] found that personality and the cognitive ability both have a decisive influence on the programming performance in the context of object-oriented programming. More interestingly is that they found in contrast to previous studies (e.g. [ES89]) that from those two factors the personality shows the stronger predictive power. Darcy and Meng [DM05] did not find a relationship between personality and programming performance and also programming experience did not correlate with the performance. However, they found that older participants and those with a better academic performance did perform better in the programming task.

[5]The *Basic Programming Knowledge Test* (BPKT) is a paper-and-pencil test used as a criterion of programming proficiency. It has been developed by Berger et al., 1966 [Rig66].

3 Coding challenges

A coding challenge, as introduced in section 1.1, is an algorithm and coding problem to assess a programmers problem-solving skills. We introduce the term *coding challenger* to describe a person who attempts to solve a coding challenge.

3.1 Areas of application

It is our impression that the term *coding challenge* is used as a synonym to *programming challenge* and nowadays more common for competitive and algorithmic programming problems than for programming or IT hurdles. Coding challenges are used in several areas of applications and on the internet there are websites that offer different types of coding challenges for learning, practicing and competing. Most often the coding challenger has to find the most efficient algorithm or any correct algorithm within limited time. These are the ones relevant to our work. Other types of coding challenges include those that can be solved by completing code to win a game or by writing code that passes all given test cases.

Coding challenges are well known for their usage in technical job interviews, especially as they are used by some of the most successful technology companies [McD15]. One can be critical about the usefulness of coding challenges for assessing a candidate's programming abilities in a technical job interview. Whiteboard coding in general is arguably far from a software engineer's daily work. However, interviewers rely on conducting coding challenges to find a particular kind of employee, not only, but particularly because attempting to solve a coding challenge demonstrates a programmers thought processes and if he or she is actually able to translate an algorithm into code [McD15]. Second, coding challenges are also used as tasks for programming competitions. In fact, programming competitions enjoy wide popularity. In 2017, the ACM International Collegiate Programming Contest (ICPC) recorded a total of 46,381 students from 2,948 universities in 103 countries [Bay17]. In the same year, the winner of the Google Code Jam prevailed against more than 25,000 competitors and won a grand prize of $15,000 [Goo17].

Finally, coding challenges are often used for attracting and teaching students. Challenging and competitive tasks motivate students. They can be used for teaching the appropriate use of data structures and the functioning of algorithms. And the characteristics of coding challenges allow for an automatic judging of assignments which results in immediate feedback for the students. We elaborated on this point in section 2.1.

3.2 Good coding challenges

The answer to what makes a good coding challenge is subjective and to the best of our knowledge there is no scientific research on this topic. For sure the selection of an appropriate coding challenge depends on the context and what the reasons for conducting or attempting the challenge are. For example, an interviewer wants to test a candidate's ability to develop an algorithm, whereas a teacher of a programming course might want to teach time and space complexity. Coding challenges will be selected accordingly. From what we know from the opinions and experiences of interviewers, we can argue that the existence of the following characteristics of a coding challenge has proven its worth in technical interviews [McD13; McD17].

A so-called brute-force solution, which describes an algorithm that systematically goes through all possible solutions to a given problem, should not be the most efficient solution to the problem. The reason is that brute-force algorithms usually are the most obvious way of solving a coding challenge and so they are the first thing that even a below-average coding challenger can come up with. If one reason for doing a coding challenge is to find out if a coding challenger can critically think about his or her initial solution and how this solution can be optimized, then coding challenges with an inefficient brute-force solution and ways to improve it are great. Again, for interviewers it is important to see the logical thinking process and how the coding challenger attempts an unknown problem [McD13]. A coding challenge should therefore also not just test a single piece of knowledge, for example, a particular programming language feature, except this is what the interviewer aims for. There would be a high chance that some otherwise good coding challengers do not know about this single fact and thus the results become unreliable. More generally, McDowell recommends interviewers to "use hard questions, not hard knowledge" in order to focus on problem-solving and other skills that can not quickly be learned at work [McD15].

As described in the later section 4.1 and discussed in section 6.2 we designed our study in a way that the coding challenger only has to interact with his computer and that there should be no need for asking for further clarification. This is different from some technical interviews where the challenge description does not give enough information to find a satisfactory solution and the candidate is expected to ask further questions

before attempting to solve the coding challenge. In our study we only measure the combination of finding and implementing an algorithm to a given coding challenge, and neither explicitly observe how well participants are in understanding problem statements nor how much a participant cares about requirements engineering. Thus for our study a good coding challenge is unambiguous and easily understandable.

3.3 Required knowledge for solving

The type of coding challenges we describe in this work requires some knowledge for successfully finding and implementing an efficient solution [McD15; MK+12]. First, without the use of appropriate data structures a coding challenger will not be able to solve most coding challenges with an efficient algorithm or will not be able to find a working solution at all. In-depth knowledge about data structures covers in particular the structure, strengths and limitations of strings, arrays, hash tables, linked lists, stacks, queues, trees and graphs. Second, there are some fundamental algorithms that every coding challenger should be aware of. For example, to know sorting and searching algorithms on common data structures as well as to know their time and space complexity is often the basis for coding challenge solutions. Understanding the concept of complexity in general is required knowledge when it comes to assessing the efficiency of an algorithm. For the harder problems, some more concepts such as dynamic programming, bit manipulation and mathematical principles must be understood, but this of course depends on the context as, for example, the finals of the Google Code Jam and a beginners' course on data structures and algorithms would have different requirements for the coding challengers.

Once a coding challenger has solved a sufficiently large number of coding challenges, he or she might be able to quickly classify a given problem and to minimize the set of potential data structures and algorithms for its solution. Otherwise, one can follow the advice to walking through a problem by McDowell [McD15], which basically is to find a brute-force solution and then optimize it by eliminating bottlenecks, unnecessary work and duplicated work.

3.4 Judging solutions

In a technical interview, interviewers compare a candidate's performance to other candidates who had to solve the same coding challenge. So it is not necessarily a disadvantage to get a hard challenge as it might be hard for every other candidate as well. The time a candidate needs to get to a correct and efficient solution and how

much guidance and help a candidate receives from the interviewer are criteria for the evaluation of the candidate [McD15].

The way solutions to coding challenges get evaluated in programming contests varies, but there are more objective and absolute judgment criteria than in a technical job interview. It is common that solutions are scored by correctness and for that purpose for each coding challenge there are a couple of test cases. However, different scoring schemes have been proposed, that differ in the way of scoring solutions which pass *some* test cases [KVC06]. For example, the ACM ICPC differentiates only between a correct and an incorrect solution. If any test case fails then a solution receives no points. The more problems a team solves, the higher it is ranked and in case of a tie, the time needed to solve the problems is the decisive criterion [BS16]. The Google Code Jam works in a similar way, except that each time a participant submits an incorrect solution he or she receives an additional time penalty. In case of a tie, the participant with the lowest penalty time will be ranked first.
Other scoring schemes give points for each successful test case. Graeme et al. [KVC06] proposed to award the full score for each batch of tests if the solution produces a correct output for *any* test case in that batch. In addition to judging the correctness of a solution, some contests also have a time limit in which a solution has to produce a correct output for a given test case [BS16]. This forces participants of a programming competition to not only find a correct algorithm but also to find an efficient one.

In section 4.5 we describe how we scored the solutions of the participants of our study. Basically, we combined the relative evaluation that is common for technical interviews with the *All-or-Nothing scoring* [KVC06] known from several programming competitions. If a given solution passed all test cases it was scored according to time complexity and relative to the other solutions. We specified a time limit for each challenge but did not consider the time to complete a challenge in our scoring scheme.

4 Methodology

In our work we want to find hypothesis for the varying performance in coding challenges among software engineers.

4.1 Research design

To answer the research question we conducted an exploratory study in which participants had to solve three coding challenges on a computer and in between they filled out questionnaires about their individual characteristics. Exploratory research intends to gain information for further research through exploring a research question. “Exploratory research cannot provide a conclusive answer to research problems [...], but they can provide significant insights to a given situation” [Sin07].

To motivate potential participants to take part in the study, they were told that the study would last at most 90 minutes and that it was about coding challenges, similar to those used by several software and technology companies during their interview process. They were also told that the reason for the study is to find out why some software engineers perform better in coding challenges than others. Per slot, one or two participants then got invited to a quiet room where they were provided with an informed consent and introduced verbally to the study. We used a script to make sure that every participant received the same information. A translated English version of this script is given in the appendix. After the introduction, participants had the chance to ask questions before they started to fill out the first questionnaire.

Participants had to solve coding challenges on a computer without the use of the internet or other helping material. To make sure that there was no advantage or disadvantage for any participant due to not knowing the used development environment, participants were asked if they were familiar with Eclipse and Java. Each coding challenge had to be solved individually and within a given time. There was a method signature given so that the type of the return value as well as the parameters of this method were defined. It was therefore not allowed to change the method signature in any way. A description of the problem was given as a comment above the method. We describe the challenges in section 4.3. The task then was to implement the method with a time-efficient solution to

the problem. It was allowed to add private methods if needed and to use methods and data structures of the predefined Java packages. Participants were told that the solutions would be evaluated by correctness and time complexity, which are common judgment criteria for modern programming contests [For10] and coding interviews [McD15]. While solving a coding challenge, participants were allowed to take notes on paper.

In addition to the given method signature for each challenge, there was also a `main` method given with an example call of the method to implement. The expected output was given as well. We provided the example to make it easier for the participants to understand the task and to increase the likelihood that no further questions were necessary while a participant solved a coding challenge. The participants were allowed to run the `main` method and to modify it to add their own test cases if desired. There was no other feedback on the correctness or efficiency of a participant's solution than what the `main` method tested. Participants were told that there was no advantage in submitting a solution before the time was up. If a candidate implemented multiple solutions within the time limit, he or she had to decide which one to submit in the end. When evaluating the solutions afterwards we only considered the implementation given in the prescribed method signature.

4.2 Participants

All participants of the study had to be software engineering students of the University of Stuttgart and they had to be at least at the end of the second semester of their bachelor's program. The reason for the latter requirement is that at this point of time a student has the fundamental knowledge of data structures, algorithms and time complexity that is required for solving the coding challenges in a time-efficient way. In their second semester, software engineering students attend a lecture which is specifically about data structures and algorithms.

The sample consisted, on the one hand, of 14 bachelor students at the end of the second semester. As part of a course they had to take part in a study and they were given the choice between five different studies, including ours. Additionally, the sample consisted of seven students who were in a higher bachelor semester and eleven students who studied in the master's program. These students were personally invited by email. In total, 32 participants took part in our study.

4.3 Coding challenges

For the study we selected three different coding challenges. As described in section 4.1 the participants had to solve them one at a time, in Eclipse and within a given time limit. In section 3.2 we discussed what makes a good coding challenge and based on this discussion we chose easily understandable challenges with several possible solutions. Additionally, we chose challenges were the finding of an efficient algorithm was challenging but the implementation should have been straightforward. To not make the participants spend too much time on handling edge cases, some limitations to the input parameters were given in the task description.
The set of coding challenges we chose covers a range of concepts which are commonly required for solving coding challenges. This includes the use of appropriate data structures, an optimization problem and recursive thinking.

In the following we describe each of the coding challenges. They were presented to the participants in German, which is their native language, for minimizing misunderstandings. The given time limit was for understanding the task, finding an algorithm and implementing the algorithm. We piloted the study with a test participant in advance to make sure that the time limit for each challenge is not too short for the participant to come up with a solution. Additionally, to control for time pressure we asked the participants after each coding challenge if they felt that they were under time pressure.

4.3.1 Challenge 1 - Pairs of Integers

```java
/**
 * In the given array, find the number of integer pairs with a given difference.
 *
 * An example is given in the main method.
 *
 * The following applies:
 *  numbers contains at least two integers
 *  numbers contains no duplicates
 *  dif >= 1
 */
public static int pairCount(int[] numbers, int dif) {
  // TODO
  return 0;
}

public static void main(String[] args) {
  int[] numbers = {1,5,3,6,8};
  int dif = 2;

  // Expected output: 3
  // The pairs with a difference of two are: {1,3} {5,3} {6,8}.
  System.out.println(pairCount(numbers, dif));
}
```

Listing 4.1: Coding challenge 1.

The fist coding challenge was considered to be the easiest one, at least when it comes to finding any correct solution to the problem. The brute force algorithm, where every possible combination of two numbers is tested in two `for`-loops, is straightforward and runs in $\mathcal{O}(n^2)$, where n is the number of integers. One has to go through all the integers of the array and search in the rest of the array for the integer that makes a pair with the current integer. Such a solution is given in Listing 4.2. By sorting the array in the beginning, one can improve the solution and do a binary search on the rest of the array when searching for the corresponding integer to build a pair with the current integer. This solution would run in $\mathcal{O}(n \cdot log(n))$, but it still can be improved by using an appropriate data structure. One has to go through all the integers again and each integer that has been visited already has to be put into a `HashSet`. When searching for the integer that makes a pair with the current integer, one now can check in constant time if this number is in the `HashSet`, so that each number has to be visited only once and

therefore the time complexity for this solution is $\mathcal{O}(n)$. A solution that uses a `HashSet` is given in Listing 4.3.

The implementations of these algorithms might vary, but the key thing is that there are several possible solutions of different time complexities to this first coding challenge. Participants had 15 minutes to solve this challenge.

```
public static int pairCount(int[] numbers, int dif) {
   int count = 0;
   for (int i = 0; i < numbers.length; i++) {
      for (int j = i+1; j < numbers.length; j++) {
         if (Math.abs(numbers[i] - numbers[j]) == dif) {
            count++;
         }
      }
   }
   return count;
}
```

Listing 4.2: A brute-force solution to coding challenge 1 in $\mathcal{O}(n^2)$.

```
public static int pairCount(int[] numbers, int dif) {
   HashSet<Integer> set = new HashSet<Integer>();
   int count = 0;
   for (int i : numbers) {
      if (set.contains(i + dif)) {
         count++;
      }
      if (set.contains(i - dif)) {
         count++;
      }
      set.add(i);
   }
   return count;
}
```

Listing 4.3: A solution to coding challenge 1 in $\mathcal{O}(n)$.

4.3.2 Challenge 2 - Mansion

```
/**
 * You want to build a mansion along a road. On the other side of the street
 * there are houses, in which a certain number of people live.
 *
 * Your mansion is as long as w houses together.
 *
 * Place your mansion in such a way that on the other side of the street as
 * many people as possible live opposite your mansion.
 * This greatest possible number should be returned by this method.
 *
 * An example is given in the main method.
 *
 * The following applies:
 *  1 <= w <= houses.length <= 100000
 *  The number of people in each house is >= 0
 */
public static int mansion(int[] houses, int w) {
  // TODO
  return 0;
}
```

Listing 4.4: Coding challenge 2.

This challenge was taken from the Australian Informatics Olympiad 2007. In the `main` method there was a detailed example given including an ASCII art that illustrated the scenario, similar to the example and illustration given in the task description of the AIO task[6]. Participants had 25 minutes to solve this challenge.

One approach of solving this coding challenge is to use a sliding window of length w that covers w houses and gets shifted along all the houses in the array. The sum of people living in these w houses is the amount of people living opposite the mansion. Each time when shifting the window, one can either calculate the sum of all houses covered by the window or simply use the last sum and subtract the people from the one house that is no longer covered by the window and adding the number of people of the one house that is now covered by the window. These two solutions differ in their time complexity. The first one runs in $\mathcal{O}(n \cdot w)$ and the second solution runs in $\mathcal{O}(n)$, where n is the number of houses.

[6] http://archive.is/E5qEG

```java
public static int pairCount(int[] houses, int w) {
  int count = 0;
  for (int i = 0; i < w; i++) {
    count += houses[i];
  }

  int lastWindow = count;
  for (int i = 1; i + w <= houses.length; i++) {
    int currentWindow = lastWindow - houses[i-1] + houses[i-1 + w];
    if (currentWindow > count) {
      count = currentWindow;
    }
    lastWindow = currentWindow;
  }

  return count;
}
```

Listing 4.5: A solution to coding challenge 2 in $\mathcal{O}(n)$.

4.3.3 Challenge 3 - Triple step

```java
/**
 * A child is climbing up a staircase with n steps, and can hop either 1 step,
 * 2 steps, or 3 steps at a time. Implement a method to count how many possible
 * ways the child can jump up the stairs.
 *
 * An example is given in the main method.
 *
 * The following applies:
 *  0 <= n <= 30
 *  if n = 0 return 1
 */
public static int countWays(int n) {
  // TODO
  return 0;
}

public static void main(String[] args) {
  int n = 3;

  // Expected output: 4
  // These are the possibilities to climb the three stairs:
  // {1,1,1}, {1,2}, {2,1}, {3}
  System.out.println(countWays(n));
}
```

Listing 4.6: Coding challenge 3.

The third coding challenge was supposed to be the hardest one in our study. It was taken from the book *Cracking the Coding Inteview* [McD15] and the ability of recursive thinking is beneficial to find a solving approach. A simple recursive solution with a time complexity of roughly $\mathcal{O}(3^n)$ is given in Listing 4.7.

```
public static int countWays(int n) {
  if (n < 0) {
    return 0;
  } else if (n == 0 || n == 1) {
    return 1;
  } else {
    return countWays(n - 1) + countWays(n - 2) + countWays(n - 3);
  }
}
```

Listing 4.7: A recursive solution to coding challenge 3 in $\mathcal{O}(3^n)$.

An alternative implementation would be a recursive depth-first search for all possible permutations starting from the bottom of the staircase. In either case the time complexity is not ideal as the same subtrees have to be calculated multiple times. For example in the recursive solution given in Listing 4.7 for n = 4 the algorithm calculates the solution for $countWays(2)$ twice. One could store the solutions to such a subproblem in an additional memory structure. This reduces the time complexity to $\mathcal{O}(n)$. As we told the participants that each solution is only evaluated by correctness and time complexity, we ignore the differences in space complexity at this point. However, with the iterative solution given in Listing 4.8 one can avoid the need for additional memory and get to a solution that runs in $\mathcal{O}(n)$ as well.

```
public static int countWays(int n) {
  int result = 1;
  int a = 1;
  int b = 0;
  int c = 0;

  for (int i = 0; i < n; i++) {
    result = a + b + c;
    c = b;
    b = a;
    a = result;
  }
  return result;
}
```

Listing 4.8: An iterative solution to coding challenge 3 in $\mathcal{O}(n)$.

Participants had 25 minutes to solve this challenge.

4.4 Candidates for predictor variables

The research question asks for individual characteristics of good coding challenge solvers. To assess the individual characteristics, each participant had to fill out a total of four questionnaires on the current mood, personality, academic performance and prior training and programming experience.

The first questionnaire asked for demographic data and the current mood. It had to be filled out before the first coding challenge. Demographic data included the name, an e-mail address for being able to contact the participant afterwards with follow-up questions, and the age and gender for describing the population sample. To assess the current mood we used the Scale of Positive and Negative Experience (SPANE) [DWT+10]. It is a validated model [RHS17] that subjectively measures how often a participant had a given feeling during the last four weeks and offers aggregation models. There are six positive and six negative affective states for which the participant had to indicate the frequency on a 5-point scale. The positive and negative states can be aggregated to a SPANE-P and SPANE-N value by summing up the indicated values on the 5-point scale. A subtraction of the SPANE-N from the SPANE-P value results in the so-called affect balance value, SPANE-B [DWT+10]. However, using the SPANE-B value is not recommended by Rahm et al. [RHS17] as it is not clear how to interpret this value and the calculation results in a loss of information on the actual extent of the positive and negative feelings. We will report results for the correlation coefficient for SPANE-B anyway as the pure difference in the frequency of positive and negative affects could be predictive for the performance in solving coding challenges.

The second questionnaire had to be filled out after the first coding challenge and was about personality and academic performance. Like in the third and fourth questionnaire the first question of these questionnaires asked the participant on an 5-point agreement scale if he or she was under time pressure when solving the previously assigned coding challenge. The participant's personality was assessed via a validated German version of the Big Five Inventory [LLA01] for measuring the five factor model of personality [Dig90]. In the five factor model five dimensions for describing personality traits exist: extraversion, agreeableness, conscientiousness, neuroticism and openness to experience. Different researchers used different terms for the five factors in the past [Dig90]. We use the previously given terms, which were coined primarily by Costa and McCrae [Dig90; MJ92]. The used inventory consists of 44 statements related to the five dimensions. Participants had to agree or disagree on these statements on a 5-point scale, which in turn is used to calculate a score for each of the five dimensions by summing up the participant's values for the corresponding statements. The following description of the five factors is a collection of the related statements, taken from the original English verison by John and Srivastava [JS99]. A person with a high score for

extraversion is talkative, outgoing, sociable, full of energy, assertive and generates a lot of enthusiasm. He or she does not tend to be quiet, reserved, or shy. Agreeableness is the extent to which someone is helpful, unselfish with others, generally trusting, considerate, kind to almost everyone, has a forgiving nature and likes to cooperate with others. The score for agreeableness gets reduced if a person tends to find fault with others, starts quarrels with others, can be cold and aloof and is sometimes rude to others. Conscientiousness describes the extend to which a person is a reliable worker, perseveres until the task is finished, does a thorough job and does things efficiently. A conscientious person is not somewhat careless, does not tend to be disorganized or to be lazy and is not easily distracted. Someone who is depressed, gets nervous easily, worries a lot, can be tense and moody will be scored high for the neuroticism factor. The score for this factor gets reduced if the person handles stress well, is relaxed, emotionally stable, remains calm in tense situations and is not easily upset. Finally, a person with a high score for openness to experience comes up with new ideas, is original, curious about many different things, ingenious, a deep thinker, sophisticated in art, music or literature, has an active imagination, values artistic and aesthetic and likes to reflect and play with ideas. Someone is not considered to be open to experience if he or she prefers work that is routine and has few artistic interests.

After the part on personality, in the second questionnaire each participant was asked whether he or she is currently enrolled in the bachelor's or master's degree program and in which semester. We wanted to know the current grade point average, as well as the grading for two specific exams on data structures, algorithms and time complexity. If the participant was enrolled in the master's program we also asked for the grade point average he or she finished the bachelor's program with.

The third questionnaire started with a question on the feeling for time pressure when solving the second coding challenge. Afterwards programming experience related questions were asked. On a 6-point frequency scale we wanted to know about the participant's experience with coding challenges in the past year. We then asked for the years of programming experience, the years of programming experience particularly with Java, and the years of experience the participant had with working in a software related company. If the participant was involved in open source projects, we asked for a corresponding online profile, for example a GitHub profile. We were also interested in the URL to a Stack Overflow profile if one was available. For both, the open source projects profile and the Stack Overflow profile, we were interested mainly in to what extent the participant was involved in the corresponding communities and in his or her corresponding contributions.

Finally, after the third and last coding challenge, the only question on the fourth questionnaire was intended again to control for time pressure when solving the latest coding challenge. The original questionnaire items are given in the appendix.

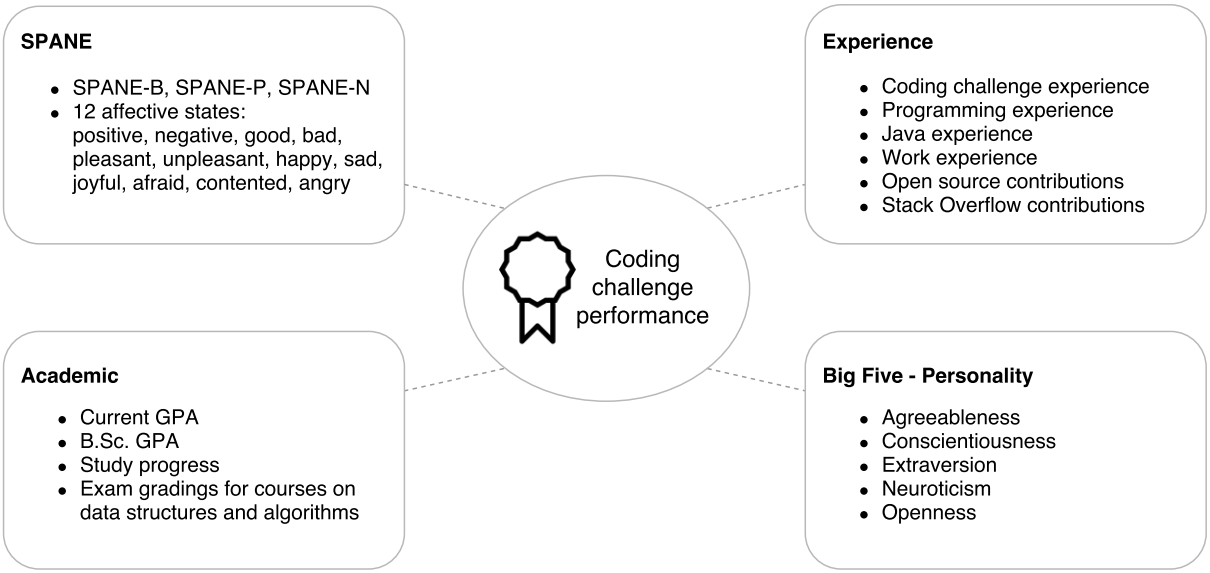

Figure 4.1: Overview of the measured variables.

4.5 Analysis procedure

To answer the research question we calculated several correlation coefficients between a participant's performance in solving the three coding challenges described in section 4.3 and the quantitative answers to the questionnaire items described in section 4.4. In order to do so, we needed to objectively score a participant's performance. Previously, in section 3.4 we described how solutions to coding challenges usually get evaluated in different areas of application. We stayed close to how recruiters evaluate the performance of candidates in technical job interviews, namely in relation to each other and with respect to correctness and time complexity. The latter aspect was told to the participants before the study. Although it would have been possible to determine the best possible time complexity class of an algorithm to the given problems, we could not be sure that this solution could be achieved under the conditions of our study. It was therefore essential to score the participant's solution in relation to each other.

For a participant's solution to a single coding challenge we first run automated test cases on the given source code to see if it produced correct results. If any test case failed, the solution was considered to be incorrect and was scored with zero points. If the solution passed all test cases we analyzed its time complexity. Solutions in the best given time complexity class, which were in our scenario the solutions with the most efficient algorithm to the problem, were scored with one point. In case participants came up with more than one correct algorithm in different complexity classes, the solutions were ranked and scored on a linear scale between zero and one. This means, for example, that for two correct solutions in different time complexity classes the second best solution

receives a score of 0.5, whereas for three correct solutions in different time complexity classes the second best receives a score of 0.66 and the third best solution receives a score of 0.33.
In addition we multiplied the achieved score for a coding challenge with a constant factor that was calculated in relation to the number of correct solutions for this coding challenge. We considered coding challenges with less correct solutions as harder and therefore multiplied the points for a correct solution with a higher factor and in such way that the worst solution to a significantly harder challenge received the same score as the best solution to the easier challenge(s). Otherwise the second best solution to the hardest problem would have been scored with less points than a solution for an easy problem that every participant came up with. In technical interviews one usually does not have to handle the issue of weighting individual coding challenges as there is only one problem that has to be solved. Some programming competitions score each coding challenge equally with one point for a correct solution or zero points for an incorrect solution. Other competitions, like the Google Code Jam, award points depending on the size of the sub-problems. A concrete scoring scheme based on our results is given in Table 5.1.

The overall performance score of a participant was obtained through summing up the scores for each of the three coding challenges.

We used Pearson's correlation coefficient to explore which individual characteristics were related to the performance in solving coding challenges. For variables that did not meet the assumptions for the Pearson's correlation coefficient we used Spearman's rank correlation coefficient. In both cases the coefficient value ranges from -1 to 1. We are aware of an open debate on the nature of frequency and agreement scales such as the ones we used for assessing the current mood (SPANE) and the personality (Big Five inventory), which affects the decision for or against Pearson's r. For the affective states we decided against Pearson's r and instead reported Spearman's rho. The distribution of our sample data for the affective states was not even close to a normal distribution and we had no reason to believe that this was due to a small sample size. The same applied for work experience. Other variables for which we calculated Spearman's rho because of their types of data were the experience with coding challenges and the study progress.

We report all calculated correlation coefficients in chapter 5 with emphasis on moderate and strong relationships.

5 Results

The sample consisted of 32 software engineering students, from which five identified themselves as female and 27 as male. The average age of the participants was 21.88 years with a standard deviation of 2.4.

The study was designed to answer the question of which individual characteristics make a good coding challenge solver. We calculated a performance score for each participant and evaluated his or her answers to our questionnaires on the current mood, personality, academic performance and prior training and experience. Identifiers for the participants reflected the performance scores descending. This contributes to the anonymization process and whenever we refer to a particular participant we know that he or she is at least as good as all participants with higher numbers in their identifier. Participants were not aware of this.

We first want to report on the performance of each individual participant to give a clear overview of the frequency of time complexity classes for correct solutions and how the overall score for each participant is achieved. In section 4.5 we described the scoring method and explained why the scores for harder challenges have to be multiplied with a higher factor than the scores for easier challenges and that the weighting for a coding challenge is based on the number of correct solutions for this coding challenge. As it was significantly harder to implement a correct solution for the third coding challenge than for the other two challenges, the factors are 1.0 for the first and the second challenge and 2.0 for the third challenge. The worst solution to the third coding challenge now receives the same score as the best solution to the easier coding challenges. A maximum score of 4.0 was achievable. Table 5.1 shows the concrete scoring scheme and Table 5.2 shows the resulting scores for the participants and illustrates the effect of our scoring scheme. However, the ranking of the participants would not look much different with equal factors of 1.0 for all three coding challenges. P06 would have the same score as P07 to P11. P02 would be ranked after P05 and before P06.

The average participant had 1.72 correct solutions and a score of 1.04. Eleven participants scored the median and mode value of 0.83. Only one participant achieved the highest score and four participants solved none of the challenges correctly and thus had a score of 0.

Challenge 1		**Challenge 2**		**Challenge 3**	
Complexity	Score	Complexity	Score	Complexity	Score
$\mathcal{O}(n)$	1.0	$\mathcal{O}(n)$	1.0	$\mathcal{O}(n)$	2.0
$\mathcal{O}(n \cdot log(n))$	0.66	$\mathcal{O}(n \cdot w)$	0.5	$\mathcal{O}(3^n)$	1.0
$\mathcal{O}(n^2)$	0.33				

Table 5.1: Mapping from a solution's time complexity class to score.

From Table 5.2 we see that 28 participants came up with a solution to the first coding challenge, but only two participants were able to implement something different from the brute-force algorithm. For the second challenge from 20 correct solutions nine run in linear time, which is the best possible for all challenges. Participant P02 as well as P23 to P28 were close to a solution for the second challenge, but did not implement the termination condition for their loop correctly which resulted in failed test cases. The third coding challenge was solved by six participants. Although the number of correct solutions was the smallest of the three coding challenges and the number of complexity classes to which these solutions belong was not higher than for the other coding challenges, with four different algorithms the diversity of correct solutions was the highest for the third challenge.

After each coding challenge participants had to indicate on a 5-point scale how much they agree on the statement that they were under time pressure when solving the previously assigned coding challenge. We mapped their answers to values from one to five, where one means that they strongly disagreed and five means that they strongly agreed. Accordingly, for the first coding challenge the average time pressure was 2.22 and 1.92 for only those participants that came up with a correct solution. For the second coding challenge the average time pressure was 2.19 and 1.71 for participants with correct solutions. The median value was 2 and the mode was 1 for both coding challenges, which means that participants most often disagreed on the statement that they felt under time pressure. This is different for the third coding challenge, for which the average time pressure was 3.84 (median = 4.5, mode = 5) and 2.67 for the six participants with a correct solution.

Participant	Challenge 1	Challenge 2	Challenge 3	Score
P01	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	4.0
P02	$\mathcal{O}(n \cdot log(n))$	–	$\mathcal{O}(n)$	2.66
P03	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(3^n)$	2.33
P04	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(3^n)$	2.33
P05	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(3^n)$	2.33
P06	$\mathcal{O}(n^2)$	$\mathcal{O}(n \cdot w)$	$\mathcal{O}(3^n)$	1.83
P07	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	–	1.33
P08	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	–	1.33
P09	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	–	1.33
P10	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	–	1.33
P11	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	–	1.33
P12	$\mathcal{O}(n^2)$	$\mathcal{O}(n \cdot w)$	–	0.83
P13	$\mathcal{O}(n^2)$	$\mathcal{O}(n \cdot w)$	–	0.83
P14	$\mathcal{O}(n^2)$	$\mathcal{O}(n \cdot w)$	–	0.83
P15	$\mathcal{O}(n^2)$	$\mathcal{O}(n \cdot w)$	–	0.83
P16	$\mathcal{O}(n^2)$	$\mathcal{O}(n \cdot w)$	–	0.83
P17	$\mathcal{O}(n^2)$	$\mathcal{O}(n \cdot w)$	–	0.83
P18	$\mathcal{O}(n^2)$	$\mathcal{O}(n \cdot w)$	–	0.83
P19	$\mathcal{O}(n^2)$	$\mathcal{O}(n \cdot w)$	–	0.83
P20	$\mathcal{O}(n^2)$	$\mathcal{O}(n \cdot w)$	–	0.83
P21	$\mathcal{O}(n^2)$	$\mathcal{O}(n \cdot w)$	–	0.83
P22	$\mathcal{O}(n^2)$	$\mathcal{O}(n \cdot w)$	–	0.83
P23	$\mathcal{O}(n^2)$	–	–	0.33
P24	$\mathcal{O}(n^2)$	–	–	0.33
P25	$\mathcal{O}(n^2)$	–	–	0.33
P26	$\mathcal{O}(n^2)$	–	–	0.33
P27	$\mathcal{O}(n^2)$	–	–	0.33
P28	$\mathcal{O}(n^2)$	–	–	0.33
P29	–	–	–	0.0
P30	–	–	–	0.0
P31	–	–	–	0.0
P32	–	–	–	0.0

Table 5.2: Individual performance of the participants. Each row contains the time complexity classes of a participant's correct solution to the corresponding challenge. Incorrect solutions are marked with a dash.

Variable	r_s	p
SPANE-B	0.055	0.770
SPANE-P	0.031	0.870
SPANE-N	-0.139	0.455
positive	-0.148	0.426
negative	0.166	0.373
good	-0.030	0.872
bad	-0.237	0.199
pleasant	0.254	0.168
unpleasant	0.013	0.945
happy	-0.149	0.425
sad	-0.390	0.030
joyful	-0.062	0.740
afraid	-0.005	0.979
contented	-0.052	0.780
angry	-0.135	0.470

Table 5.3: Summary of the correlation coefficients between Scale of Positive and Negative Experience (SPANE) and the coding challenge performance score (n = 31)[7].

The Scale of Positive and Negative Experience (SPANE) consists of six positive and six negative affective states for which the participants had to indicate on a 5-point scale how often they felt this way in the past four weeks. SPANE-P is the sum of positive states and SPANE-N the sum of negative states, both are in the range between six and 30. For our participants the average SPANE-P value of 22.65 was higher than the average SPANE-N value of 12.10 and each of the six positive states that contribute to the SPANE-P value were on average higher than their counterparts.

Our results show that there is a significant moderate negative relationship between *sad* and the coding challenge performance score, $r_s(29) = -0.390$, $p < 0.05$. With reference to the discussion in section 4.5 and for the sake of completeness, for this affective state $r(29) = -0.383$. Figure 5.1 illustrates the relationship. As higher values for the affective states stand for higher frequencies, this negative correlation means that participants who were often sad in the past four weeks tended to perform worse. The Spearman's rank correlation between *bad* and the performance score is weak negative,

[7] One participant did not indicate the frequency for all of the affective states. For this reason we dropped his or her record.

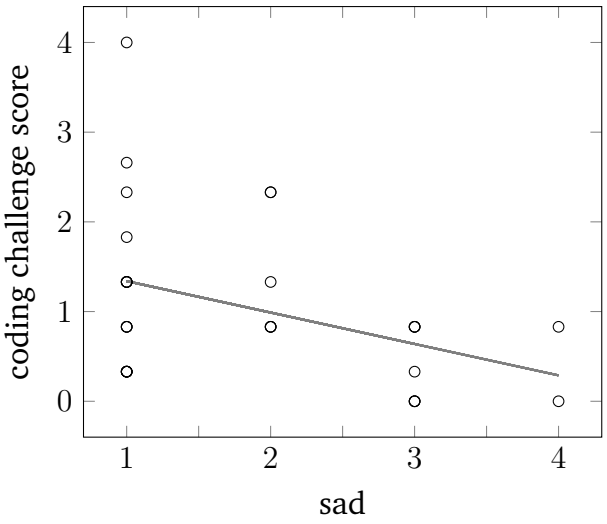

Figure 5.1: Relationship between scores for the coding challenge performance and the SPANE feeling *sad*, $r_s(29) = -0.390$, $p < 0.05$.

$r_s(29) = -0.237$, and additionally there is a weak positive relationship between *pleasant* and the performance score, $r_s(29) = 0.254$. The p-values for these two correlations are above the significance level of 0.05.

For assessing a participant's personality we used the five factor model which describes a personality with five different traits: extraversion, agreeableness, conscientiousness, neuroticism and openness. Table 5.4 shows the correlation coefficients between these traits and the coding challenge performance score. What we see from our results is that there is a significant moderate negative relationship between *conscientiousness* and the performance score, $r(30) = -0.352$, $p < 0.05$. This relationship is illustrated in Figure 5.2. Also there is a weak negative relationship between extraversion and the performance score, $r(30) = -0.228$. Conscientiousness describes the extent to which a person is a reliable worker and how easily he or she is distractible. Extraversion describes how sociable a person is. We described the five factors in more detail in section 4.4.
For the other three personality traits there is no relationship in our data.

The variables for the academic performance provide the highest values for Pearson's r in our data set. From the results shown in Table 5.5 we see that there is a strong negative relationship between the performance scores and the grade point averages master students received in their bachelor's degree, $r(8) = -0.620$. Also there is a significant strong negative relationship between the performance scores and the grades for the data structures and algorithms course, $r(16) = -0.557$, $p < 0.05$ and a significant moderate negative relationship between the performance scores and the current grade point average, $r(29) = -0.448$, $p < 0.05$. As in Germany lower grades are better than

Variable	Pearson's r	p
extraversion	-0.228	0.210
agreeableness	-0.058	0.755
conscientiousness	-0.352	0.048
neuroticism	-0.098	0.592
openness	0.069	0.708

Table 5.4: Summary of the correlation coefficients between the Big Five personality traits and the coding challenge performance score (n = 32).

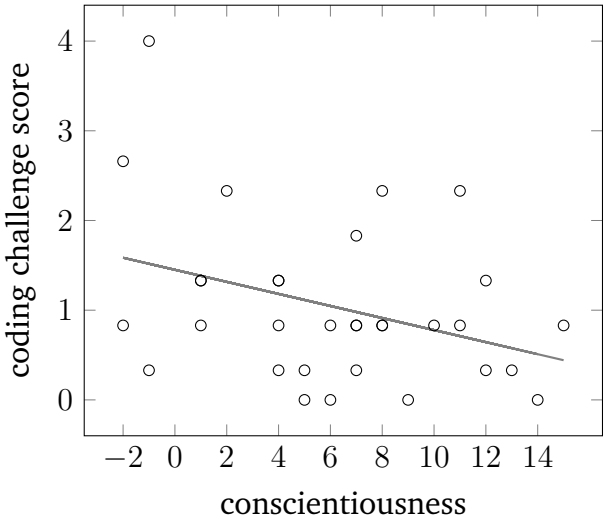

Figure 5.2: Relationship between scores for the coding challenge performance and Big Five personality trait *conscientiousness*, r(30) = -0.352, p < 0.05.

Variable	Pearson's r	r_s	p	n
Current GPA	-0.448		0.011	31
B.Sc. GPA (master students only)	-0.620		0.056	10
Grade for *data structures and algorithms* course	-0.557		0.016	18
Grade for *algorithms and complexity* course	-0.183		0.452	19
Study progress		0.179	0.328	32

Table 5.5: Summary of the correlation coefficients between the academic performance and the coding challenge performance score.

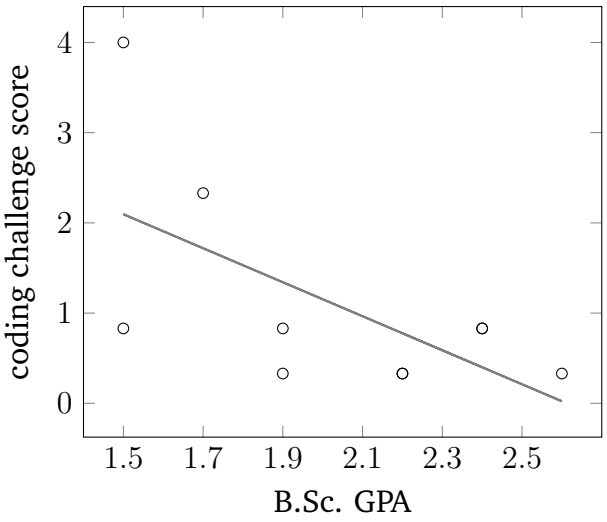

Figure 5.3: Relationship between scores for the coding challenge performance and the B.Sc. GPA of master students only, r(8) = -0.620, p = 0.056.

high grades, these negative relationships mean that participants with a better grading were also the better coding challengers.

The average grade for the data structures and algorithms course was 1.66 (sd = 0.79) and therefore much better than the average grade for the algorithms and complexity course which was 3.11 (sd = 0.83). We only see a weak negative correlation for the latter course with the coding challenge performance score, r(17) = -0.183.

A weak positive relationship can be seen between the study progress and the performance scores, $r_s(30) = 0.179$. Study progress was mapped from one of three groups: students at the beginning of their bachelor's program, students that are at least in the fourth semester of the bachelor's program and one group for the master students. Our sample consisted of 14 bachelor students that belong to the first group, seven bachelor students that belong to the second group and eleven master students that belong to the third group. The positive relationship therefore means that the more advanced students performed better, but from the correlation coefficient alone we can not tell to what extent. The highest score of 4.0 was achieved by a master student, the maximum score in the group with the advanced bachelor students was 2.66 and the maximum score for the students at the beginning of their bachelor's program was 1.83. On average the advanced bachelor students had a score of 1.89 (median = 1.83) and outperformed the first group which had an average score of 0.66 (median = 0.83) as well as the third group which had an average score of 1.07 (median = 0.83).

In the last part of the questionnaires we asked the participants experience-related questions. Programming experience, experience with Java and experience with working

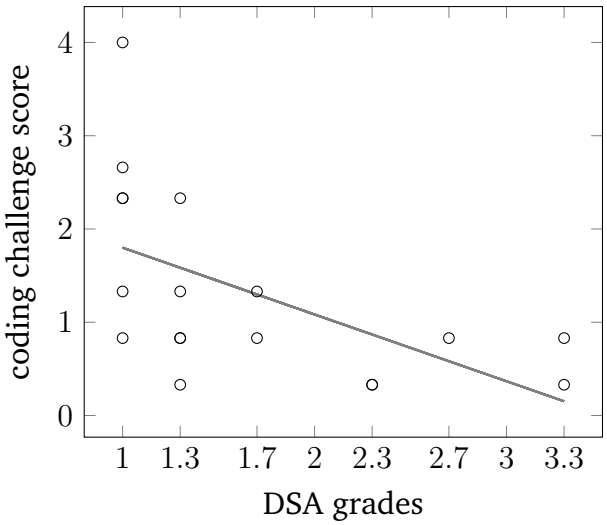

Figure 5.4: Relationship between scores for the coding challenge performance and the DSA course grades, r(16) = -0.557, p < 0.05.

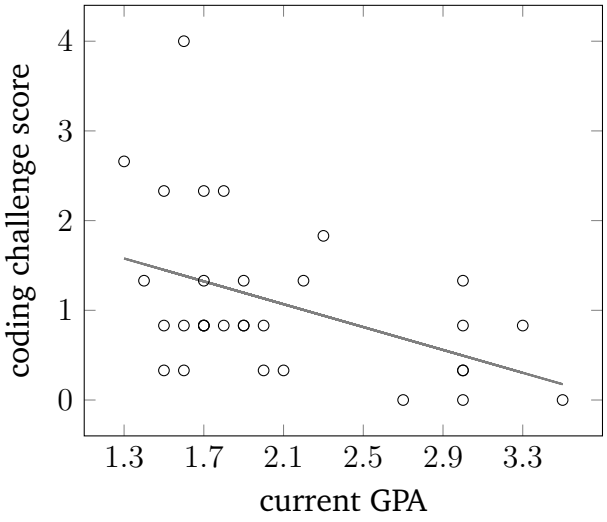

Figure 5.5: Relationship between scores for the coding challenge performance and the current GPA, r(29) = -0.448, p < 0.05.

Variable	Pearson's r	r_s	p
Coding challenge experience		0.227	0.211
Programming experience	0.420		0.017
Java experience	0.184		0.315
Work experience		0.196	0.283

Table 5.6: Summary of the correlation coefficients between experience and the coding challenge performance score (n = 32).

in a company with focus on software development were measured in years. Experience with coding challenges in the past year was indicated by the participants on a 6-point frequency scale.

From Table 5.6 we see a significant moderate positive relationship between the coding challenge performance score and the programming experience of a participant, $r(30) = 0.420$, $p < 0.05$. This relationship is illustrated in Figure 5.6. On average, participants had 5.02 years of programming experience (sd = 2.47). For the work experience, coding challenge experience and the experience with Java we only observed weak positive relationships with the performance score. 21 participants answered that they never made experience with coding challenges in the last year, five participants did coding challenges for one or two times per semester, five participants for one or two times per month and one made experience with coding challenges once per week. When we asked the participants afterwards what their concrete experience with coding challenges was, they mainly told us about exercises they had to do for the algorithm and complexity class and that these exercises were pretty similar to the coding challenges we used in our study. The one participant who indicated to solve coding challenges once per week told us that he or she makes them for fun on the internet. This participant had the second best coding challenge performance score of 2.66, while the participant with the best performance score of 4.0 never made experience with coding challenges in the last year.

As described in section 4.4 we asked participants for their online open source profiles and their Stack Overflow profile to explore the contributions to the respective communities and see how they correlate with the scores for the coding challenge performance. From the 32 participants only four participants had a Stack Overflow profile, of whom three have contributed at least one question or answer to the network. The coding challenge performance scores for these three participants were above average, but their contributions were made mainly to fields unrelated to Java, algorithms or programming puzzles. More participants provided us a URL to their GitHub or GitLab profiles and eight of them contributed at least one public pull request, but there was no relationship

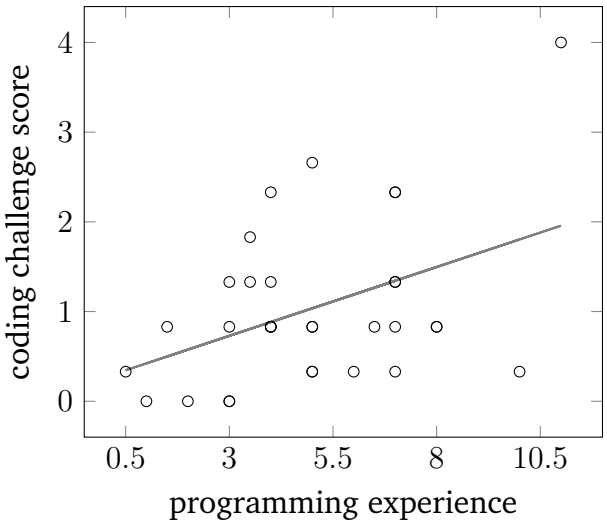

Figure 5.6: Relationship between scores for the coding challenge performance and the programming experience in years, $r(30) = 0.420$, $p < 0.05$.

between the number of pull requests and the performance score. The eight participants with public open source contributions had an average performance score of 1.16, which is slightly higher than the average performance score of 0.98 for participants without public open source contributions. The majority of projects they contributed to were programmed in Java or JavaScript.

6 Discussion

6.1 Findings

Although there exists research on different aspects of coding challenges, little research was done on what individual characteristics make a good coding challenge solver. What was known is that some software engineers perform better than others and this is also what we observed in our exploratory study. Additionally, we found relationships between some of the measured variables and the coding challenge performance score. Based on these findings we state and discuss the following hypothesis.

Hypothesis: persons with a good GPA are also good at solving coding challenges

We found moderate to strong linear correlations between two GPA-related variables and the performance score. The significant moderate negative relationship between the current GPA and the performance score, $r(29) = -0.448$, $p < 0.05$, shows that students with better grades performed better in the coding challenges. Additionally, the Pearson's correlation coefficient between the B.Sc. GPA and the performance score was strong negative, $r(8) = -0.620$, but due to the small number of master students, the relationship could have occurred by chance ($p = 0.056$).

Many of the bachelor students at the end of the second semester mentioned that their current GPA consisted only of one or two grades. As this group of students made up about 44% of our sample this should be taken into account. However, because we observed more negative relationships for grade related variables, it can be reasonably assumed that we would have observed a negative relationship also if students had taken more than one or two exams.

There was only a weak positive correlation between study progress and the performance score, $r_s(30) = 0.179$, and we observed that the group of higher bachelor semesters performed best. In discussions after the study, participants told us that in the *algorithms and complexity* course, students nowadays have to solve tasks similar to the coding

challenges we used in our study. As this course is usually taken in a higher bachelor semester, this could have been the reason for them to perform better. Furthermore, as students at the beginning of their bachelor's program performed worst and master students did not perform much worse than students in a higher bachelor semester, we assume that there is a baseline of knowledge one has to be aware of when solving coding challenges but further academic progress does not make a coding challenger better.

Taking all this, and especially the previous paragraph, into consideration, we can not assume that receiving better grades leads to better coding challenge performances. But we can assume that there is a third variable that predicts how well a student performs in his or her exams and how well he or she performs in solving coding challenges.

Hypothesis: persons with a good grade in the data structure and algorithm course are also good at solving coding challenges

Our results show a significant strong negative relationship between the grade for the *data structure and algorithms* course and the performance score, $r(16) = -0.557$, $p < 0.05$. For the weak negative relationship between the grade for the *algorithms and complexity* course and the performance score, $r(17) = -0.183$, we would like to notice that some of the participants were examined by a different examiner and they told us that the exam became therefore significantly easier.

In section 3.3 we discussed that a good understanding of data structures and algorithms is fundamental for finding an efficient algorithm to a given coding challenge. Therefore, we assume that a good preparation for the *data structure and algorithms* exam not only leads to a good grade but also improves the coding challenge performance. Taking this finding one step further, it provides at least an indication of how the targeted preparation for solving coding challenges could have an impact on the coding challenge performance.

Hypothesis: if a software engineer is sad, he or she will perform worse in coding challenges

On a more abstract level of algorithm design and execution, Graziotin et al. [GWA14] found that happy software developers perform significantly better in analytic problem solving. Although we observed a weak positive relationship between the positive affective state *pleasant* and the performance, $r_s(29) = 0.254$, we neither can confirm that there is a positive correlation between SPANE-B and the performance, nor did we find a positive

correlation between the affective state *happy* and the performance in solving coding challenges, $r_s(29) = -0.149$.

However, we found that sad software engineers performed significantly worse, $r_s(29) = -0.390$, $p < 0.05$. We believe that there could be a cause-effect relationship between sadness and the coding challenge performance, because from the participants who felt sometimes or often sad in the past four weeks (∼28%), nobody came up with a correct solution to coding challenge 3 and for the first two challenges nobody came up with an algorithm that was different from brute-force solutions. Consequently, none of the sad participants had a score higher than 0.83.

Hypothesis: gaining programming experience leads to a better coding challenge performance

For the experience-related variables we observed a weak positive relationship between the coding challenge experience and the performance, $r_s(30) = 0.227$, between the Java experience and the performance, $r(30) = 0.184$, as well as between the work experience and the performance, $r_s(30) = 0.196$. The weak correlation coefficient for the Java experience could be explained by our selection of coding challenges which do not require specific knowledge about the programming language. The positive relationship between the experience with coding challenges and the performance in solving such, is in line with what related work found (see subsection 2.1.1).

More importantly, we observed a significant moderate linear relationship between the years of programming experience and the coding challenge performance, $r(30) = 0.420$, $p < 0.05$. A study conducted by DeMarco and Lister [DL13] did not find a correlation between the years of programming experience and the performance in terms of time to complete a programming task, at least not for participants with more than six months' experience. The contradictory results could be due to the difference in the programming tasks and the different definitions of performance. Working as fast as possible on an ordinary programming task arguably requires different skills than finding an efficient algorithm to a coding challenge. According to our results we believe that an increase in programming experience leads to a better coding challenge performance. This might only hold true until a threshold, but it seems to be greater than the six months observed by DeMarco and Lister.

Hypothesis: conscientious persons perform worse in coding challenges

We expected to find that persons with a low score for *extraversion* perform worse in algorithmic problem solving and observed this correlation to a certain extent, $r(30) = -0.228$, but the results were not significant. For the other personality traits of the five factor model, we did not expect to find any correlation with the coding challenge performance and at least for *agreeableness*, *neuroticism* and *openness* we were right. However, *conscientiousness* showed a significant moderate negative relationship with the coding challenge performance score, $r(30) = -0.352$, $p < 0.05$. This means the higher the score for *conscientiousness*, the lower the coding challenge performance score. To understand this relationship we had a look at the answers to the statements of the Big Five inventory of the six participants with the highest performance score. Interestingly, they tended to be reliable workers, which increased their score for *conscientiousness* and they did not tended to be easily distracted, which would have lowered their score for the personality trait. Their average score for *conscientiousness* was 4.17, which is only slightly below the average value of 6.69 for the rest of the sample. Beside Pearson's r, we also always had a look at Spearman's rho to avoid wrong assumptions due to the possibly strong influence of outliers and sequences which are not entirely homoscedastic. Still there was a negative monotonic relationship, $r_s(30) = -0.294$. Although we can not explain the relationship, based on our findings we would like to state the hypothesis that conscientious persons perform worse in coding challenges and let future research examine this relationship in more detail.

6.2 Limitations

The study design, as described in section 4.1, has some limitations which have to be considered when interpreting the results. These limitations primarily address the external validity of our study, with respect to transferring our findings to technical interviews. Future research can benefit from our thoughts and might be able to adapt the study design accordingly.

6.2.1 An IDE and a white board differ

In a technical on-site interview candidates would rather have to solve a coding challenge on a white board than on a computer [McD15] and there are differences in the way a programmer writes code in the respective cases. An integrated development environment offers features such as syntax highlighting and automatic source code completion. It is also possible to run and debug the program on a computer within a short time to

test the correctness, compared to manually going through a test case which can also be error-prone. Syntactic errors that make a participant's solution no longer executable result in zero points in our study whereas in white board interviews, if at all, the extent to which a candidate makes these errors is considered.

Our main reasons for not conducting the coding challenges on a white board were that we are not trained as interviewers and objective scoring of a participant's solution would have been much harder if variable extent of guidance and, for example, empathy were involved. An experienced interviewer could solve this issue.

6.2.2 There was no guidance for the participants

As previously said we did not conduct white board challenges as we are not trained for conducting interviews and the variable extent of guidance would make an objective scoring of the solutions harder. However, the fact that there was no guidance for the participants was a limitation in itself. An interviewer would give small hints to the interviewee when it is appropriate [McD15]. As we did not give any hints while the participants worked on their solution for the coding challenges, it occurred, for example, that a participant came up with an overall correct algorithm but overlooked a minor mistake in its implementation that did not pass our automated test cases for the correctness of the solution. Another point is that in our study a common strategy for solving the coding challenges was to implement a working, but obvious and inefficient algorithm first and then to try to find a more efficient one, which sometimes resulted in time pressure at the end. This was only due to the fact that a participant had no other way to communicate his or her ability to find the inefficient algorithm and did not want to take the risk of not having any correct solution in the end. An interviewer who also evaluates the participants performance in the end can help to overcome this limitation.

6.2.3 Participants were unprepared

Participants of our study were students, of whom most did not prepare for solving coding challenges in the past. Not much research has been done yet to examine the effect of targeted preparation for coding challenges so we do not know about its impact and how well our results can be transferred to a group of software engineers, who all recently prepared for programming interviews or programming competitions.

Solving the challenges we chose does not require knowledge of particular programming language features or the like that would not be present to a student who is currently

enrolled in a software engineering course of study. However, refreshing one's knowledge on data structures and algorithms before solving coding challenges can be beneficial and it could improve the performance of one participant more than the performance of another participant, thus changing the correlation results. As we do not know about the impact of preparation, we at least controlled for it in our study with a question on past experience with coding challenges. Future research can repeat our study with prepared participants and see if and how the results differ.

6.2.4 Code quality did not matter

We scored the combination of finding and implementing an algorithm, which is similar to what candidates in a programming interview would have to do. While in our study we did not consider the code quality of the participants' solutions, it is relevant to a certain extent when programming in an interview. McDowell [McD15] lists the following properties for *good code* as employers want to see it: correct, efficient, simple, readable and maintainable.

While we looked at the first two properties, correctness and efficiency, we did not consider simplicity, readability, maintainability or other code quality characteristics when we evaluated the solutions. We chose problems to which possible solutions usually should be around ten to twenty lines long. Therefore, we did not discover too many differences in the solutions of different participants regarding the code quality. Also as for some of these properties like efficiency and maintainability there could be a trade-off that influences possible outcomes, we encouraged participants to focus on correctness and efficiency so that our evaluation process was as objective as possible in the end.

6.2.5 Motivation, stress and anxiety

Although some students might have had a high motivation for solving the coding challenges as an end in itself, others might have had insufficient motivation due to the fact that we did not reward them in any way. We assumed that when students voluntarily take part in our study there is already some willingness to support our research and thus participants try to get the most out of it. However, as some of our participants had to take part in a study as an exam precondition, there is still a chance that they were less motivated to try their best at the coding challenges. In future studies public and possibly anonymized score rankings of every participant's performance can arouse the ambition of the individual and give the participant a way to see how he or she performed compared to others.

Another thing to consider are emotions like stress and anxiety which are higher in a job interview where the interviewee is not anonymous, but instead facing the interviewer and trying to not fail as the stakes might be high. In our study we tried to create a relaxed atmosphere and so should an interviewer do. Future work can examine the performance of candidates in real technical interviews and assess the individual characteristics of the candidates. This would result in more meaningful results for technical interview scenarios regarding the emotional situation and also regarding some of the previously named limitations.

6.2.6 We had little knowledge on which solutions were achievable

It would have been beneficial to have evidence that participants will come up with correct and different solutions to each of the coding challenges within the parameters of our study *before* we conducted the study. In the worst case we would have ended up with everybody receiving the same score, as, for example, nobody was able to solve any challenge correctly or everybody was able to solve every coding challenge in the best possible way. To counteract this limitation, we first selected coding challenges where we knew that there were different solutions and the best possible solution was not the most obvious one. We then piloted the study with one test participant and had a good feeling about the outcome as the test participant solved the first challenge with the most obvious but not the most efficient solution, the second coding challenge with the best possible solution and for the third coding challenge the test participant could not come up with a correct solution but was able to understand possible solutions that we explained in a subsequent talk.

From our results we see that the selected coding challenges were a good choice in terms of difficulty, because there is a wide variety of well distributed scores. We also see that at least for the first two coding challenges participants were not under time pressure. For the third coding challenge we do not know if participants, who did not solve the challenge correctly, would find a correct solution if they had more time.

Future studies could use more test participants to have greater assurance that the solutions to the coding challenges will vary between participants and tweak their selection accordingly before conducting the study.

7 Conclusion

The ability to be a successful coding challenge solver is essential in programming competitions and many technical job interviews. We found that a good coding challenger is an experienced programmer, is not conscientious, was seldom sad in the past weeks, has a good GPA and a good grade in the data structures and algorithms course. For moral reasons, we do not want to suggest students to be less conscientious, if this is possible at all. To be less often sad in the time before attempting to solve coding challenges in competitions or technical interviews is advisable, but this might not always be within one's power.
As discussed in the previous chapter, we do not believe that there is a cause-effect relationship between the GPA and the coding challenge performance. However, we might be wrong in this point and even if there is no cause-effect relationship, receiving good grades would still be beneficial in other areas of life. For hiring managers the found relationship between the academic performance and the performance in solving coding challenges can be taken into consideration when having to review a lot of applications and to decide which candidate to invite to a technical interview, although we would recommend to not only focus on grades for evaluating a candidate.
Finally, what a software engineer can do to possibly improve his or her coding challenge performance is to gain programming experience and prepare well for the exam on data structures and algorithms. We believe that up to a certain threshold gaining programming experience leads to a better performance and that knowledge attained in the data structure and algorithms course has a positive influence on the coding challenge performance.

Future work

Future work can statistically confirm or refute our hypothesis. Taking some of the limitations into consideration, future studies can be designed to be more similar to technical interviews and could conduct such a study with prepared candidates to see how the results differ.
From the set of hypothesis, we would find it particularly interesting to examine the relationship between the academic performance and the coding challenge performance.

In case that there is a third variable that influences both the GPA and the coding challenge performance, it might be worth to investigate this relationship and to find ways to improve in this third variable. It would also be interesting to find out if there is a cause-effect relationship between programming experience and the coding challenge performance and if there is a threshold for which this relationship can be observed.

A Script for introducing participants into the study

For the verbal introduction of participants to the study we used a script to make sure that every participant received the same information. The following is a translated English version of our notes that we read to the participants after a cordial welcome and after they have read the consent form.

- Do you have questions regarding the consent form?
- It is important to understand that I will be able to see from who the answers to the questionnaires and the solutions to the coding challenges are. However, before I discuss the results with anybody, including my supervisor and examiner, these data will be anonymized.
- Do you study software engineering?
- Are you familiar with Eclipse and Java?
- There will be three coding challenges in total and between them you have to fill out questionnaires. At the end of each questionnaire there is the instruction to raise your hand. I will then open the next coding challenge for you.
- For each coding challenge a method signature is given. It is not allowed to change this signature, so that the type of the return value and the parameters of the methods are well-defined. Above the method signature, there is a comment explaining what the task is.
- It is allowed to create private methods if this helps you to structure your solution or to solve the problem.
- It is not allowed to use the internet when solving the coding challenges. For answering the questionnaires it is allowed to use the internet. You will be asked, for example, for your DSA mark. If you do not remember it, it is fine to look on the internet.
- It is allowed to use all methods and data structures from the standard Java package ($import java.*$).

- It is allowed to run the code. With each challenge there is also a `main` method given which contains a call of the method that you have to implement. For the input of that call the expected output is given as well. You are allowed to modify the `main` method if desired.
- You have a pen and a paper at your place for taking notes.
- For each challenge there is a time limit for understanding the problem, finding an algorithm and implementing the algorithm. For example, for the first coding challenge this time limit is 15 minutes. If you want to submit a solution before the time is up, please tell me and we will continue. This will not affect the score for your solution, so it might be a good idea to use the remaining time in order to improve your algorithm.
- We evaluate your solutions for correctness and time complexity. It is always better to have a correct solution than no solution. And the more efficient the solution is in terms of the Big O notation, the better the score.
- Please do not use your mobile phone in the next 90 minutes.
- Do you have any questions?

B Questionnaire items

In section 4.4 we describe four questionnaires on the current mood, personality, academic performance and prior training and programming experience that had to be filled out by the participants of our study. In the following the original questionnaire items are given.

Demographische Fragen

(1.) Vor- und Nachname:

(2.) E-Mail-Adresse:

(3.) Alter:

(4.) Geschlecht, mit dem Sie sich identifizieren:
[ ] männlich
[ ] weiblich
[ ] anderes, und zwar:

Aktuelle Gefühlslage

(5.) Bitte denken Sie an das, was Sie in den letzten 4 Wochen getan und erlebt haben. Anschließend kreuzen Sie bitte in der folgenden Liste an, wie häufig Sie sich so gefühlt haben.

	sehr selten oder nie	selten	gelegentlich	oft	sehr oft od. immer
positiv	☐	☐	☐	☐	☐
negativ	☐	☐	☐	☐	☐
gut	☐	☐	☐	☐	☐
schlecht	☐	☐	☐	☐	☐
angenehm	☐	☐	☐	☐	☐
unangenehm	☐	☐	☐	☐	☐
glücklich	☐	☐	☐	☐	☐
traurig	☐	☐	☐	☐	☐
von Freude erfüllt	☐	☐	☐	☐	☐
ängstlich	☐	☐	☐	☐	☐
zufrieden	☐	☐	☐	☐	☐
wütend	☐	☐	☐	☐	☐

Frage zu Coding challenge #1

(6.) Ich bin bei Coding challenge #1 unter Zeitdruck geraten
[ ] stimme voll und ganz zu
[ ] stimme eher zu
[ ] teils/teils
[ ] stimme eher nicht zu
[ ] stimme gar nicht zu

Persönlichkeit

(7.) Im Folgenden finden Sie eine Reihe von Beschreibungen, die auf Sie zutreffen können oder nicht. Bitte schreiben Sie eine Zahl links neben jede der aufgeführten Beschreibungen, um anzuzeigen, wie sehr diese Aussage auf Sie zutrifft oder nicht zutrifft.

Trifft überhaupt nicht zu	Trifft wenig zu	Trifft teils/teils zu	Trifft gut zu	Trifft sehr gut zu
0	1	2	3	4

Ich sehe mich selbst als jemand, der ...

	1. gesprächig ist, sich gerne unterhält		23. bequem ist und zur Faulheit neigt
	2. dazu neigt, andere zu kritisieren		24. ausgeglichen ist, nicht leicht aus der Fassung zu bringen
	3. Aufgaben gründlich erledigt		25. erfinderisch und einfallsreich ist
	4. deprimiert, niedergeschlagen ist		26. durchsetzungsfähig und energisch ist
	5. originell ist, neue Ideen entwickelt		27. sich kalt und distanziert verhalten kann
	6. eher zurückhaltend und reserviert ist		28. nicht aufgibt ehe die Aufgabe erledigt ist
	7. hilfsbereit und selbstlos gegenüber anderen ist		29. launisch sein kann, schwankende Stimmungen hat
	8. etwas achtlos sein kann		30. künstlerische und ästhetische Eindrücke schätzt
	9. entspannt ist, sich durch Stress nicht aus der Ruhe bringen lässt		31. manchmal schüchtern und gehemmt ist
	10. vielseitig interessiert ist		32. rücksichtsvoll und einfühlsam zu anderen ist
	11. voller Energie und Tatendrang ist		33. tüchtig ist und flott arbeitet
	12. häufig in Streitereien verwickelt ist		34. ruhig bleibt, selbst in angespannten Situationen
	13. zuverlässig und gewissenhaft arbeitet		35. routinemäßige und einfache Aufgaben bevorzugt
	14. leicht angespannt reagiert		36. aus sich herausgeht, gesellig ist
	15. tiefsinnig ist, gerne über Sachen nachdenkt		37. schroff und abweisend zu anderen sein kann
	16. begeisterungsfähig ist und andere mitreißen kann		38. Pläne macht und diese auch durchführt

Trifft überhaupt nicht zu	Trifft wenig zu	Trifft teils/teils zu	Trifft gut zu	Trifft sehr gut zu
0	1	2	3	4

	17. nicht nachtragend ist, anderen leicht vergibt		39. leicht nervös und unsicher wird
	18. dazu neigt, unordentlich zu sein		40. gerne Überlegungen anstellt, mit Ideen spielt
	19. sich viele Sorgen macht		41. nur wenig künstlerische Interessen hat
	20. eine lebhafte Vorstellungskraft hat, phantasievoll ist		42. sich kooperativ verhält, Zusammenarbeit dem Wettbewerb vorzieht
	21. eher still und wortkarg ist		43. leicht ablenkbar ist, nicht bei der Sache bleibt
	22. anderen Vertrauen schenkt		44. sich gut in Musik, Kunst und Literatur auskennt

Bisherige akademische Leistung

(8.) Ich studiere im
[ ] Bachelor
[ ] Master

(9.) Im wievielten Fachsemester befinden Sie sich?

(10.) Aktueller Notenschnitt:

(11.) Falls Sie sich im Master befinden, geben Sie bitte hier zusätzlich den Bachelor-Notenschnitt an:

(12.) Falls Sie die Prüfung „Datenstrukturen und Algorithmen“ abgelegt und bestanden haben, geben Sie bitte Ihre Note an:

(13.) Falls Sie die Prüfung „Algorithmen und Berechenbarkeit“ abgelegt und bestanden haben, geben Sie bitte Ihre Note an:

Frage zu Coding challenge #2

(14.) Ich bin bei Coding challenge #2 unter Zeitdruck geraten
[ ] stimme voll und ganz zu
[ ] stimme eher zu
[ ] teils/teils
[ ] stimme eher nicht zu
[ ] stimme gar nicht zu

Programmiererfahrung

(15.) Wie häufig haben Sie im letzten Jahr Erfahrungen mit Coding challenges gemacht?
[ ] Nie
[ ] 1-2 mal pro Semester
[ ] 1-2 mal pro Monat
[ ] Einmal pro Woche
[ ] 2-3 mal pro Woche
[ ] Täglich

(16.) Programmiererfahrung in Jahren:

(17.) Erfahrung mit Java in Jahren

(18.) Arbeitserfahrung mit Tätigkeitsschwerpunkt auf Softwareentwicklung in Unternehmen in Jahren:

(19.) Falls Sie an Open-Source-Projekten mitarbeiten, geben Sie bitte einen Link zu Ihrem Profil an (z.B. GitHub):

(20.) Falls Sie einen „Stack Overflow“-Account besitzen, geben Sie bitte einen Link zu Ihrem Profil an:

Frage zu Coding challenge #3

(21.) Ich bin bei Coding challenge #3 unter Zeitdruck geraten
[ ] stimme voll und ganz zu
[ ] stimme eher zu
[ ] teils/teils
[ ] stimme eher nicht zu
[ ] stimme gar nicht zu

Bibliography

[AI10] B. Alexander, C. Izu. “Engaging weak programmers in problem solving.” In: *Education Engineering (EDUCON), 2010 IEEE*. IEEE. 2010, pp. 997–1005 (cit. on p. 18).

[ALP12] A. Aziz, T.-H. Lee, A. Prakash. *Elements of Programming Interviews: The Insiders' Guide*. EPI, 2012 (cit. on p. 18).

[Are06] A. S. Arefin. “Art of Programming Contest.” In: *C Programming Tutorials, Data Struc-tures and Algorithms. ISBN* (2006), pp. 984–32 (cit. on p. 18).

[Ast04] O. L. Astrachan. “Non-competitive programming contest problems as the basis for just-in-time teaching.” In: *Frontiers in Education, 2004. FIE 2004. 34th Annual*. IEEE. 2004, T3H–20 (cit. on p. 19).

[Bay17] Baylor University. *ICPC History - The 2017 World Champions*. 2017. URL: https://web.archive.org/web/20171027104851/https://icpc.baylor.edu/community/history-icpc-2017 (cit. on p. 23).

[BH08] B. A. Burton, M. Hiron. “Creating informatics olympiad tasks: exploring the black art.” In: *Olympiads in Informatics* 2 (2008), pp. 16–36 (cit. on p. 17).

[BS16] A. Bloomfield, B. Sotomayor. “A Programming Contest Strategy Guide.” In: *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. ACM. 2016, pp. 609–614 (cit. on pp. 18, 19, 26).

[CH06] C. G. Cegielski, D. J. Hall. “What makes a good programmer?” In: *Communications of the ACM* 49.10 (2006), pp. 73–75 (cit. on p. 21).

[CH17] N. Cheng, B. Harrington. “The Code Mangler: Evaluating Coding Ability Without Writing any Code.” In: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM. 2017, pp. 123–128 (cit. on p. 19).

[CJW11] D. Coles, C. Jones, E. Wynters. “Programming contests for assessing problem-solving ability.” In: *Journal of Computing Sciences in Colleges* 26.3 (2011), pp. 28–35 (cit. on pp. 19, 20).

[CKLO03] B. Cheang, A. Kurnia, A. Lim, W.-C. Oon. “On automated grading of programming assignments in an academic institution.” In: *Computers & Education* 41.2 (2003), pp. 121–131 (cit. on pp. 19, 20).

[CSC15] S. Cruz, F. Q. da Silva, L. F. Capretz. “Forty years of research on personality in software engineering: A mapping study.” In: *Computers in Human Behavior* 46 (2015), pp. 94–113 (cit. on p. 20).

[Dag10] V. Dagienė. “Sustaining informatics education by contests.” In: *International Conference on Informatics in Secondary Schools-Evolution and Perspectives*. Springer. 2010, pp. 1–12 (cit. on p. 18).

[DF08] V. Dagienė, G. Futschek. “Bebras international contest on informatics and computer literacy: Criteria for good tasks.” In: *International Conference on Informatics in Secondary Schools-Evolution and Perspectives*. Springer. 2008, pp. 19–30 (cit. on p. 17).

[Dig90] J. M. Digman. “Personality structure: Emergence of the five-factor model.” In: *Annual review of psychology* 41.1 (1990), pp. 417–440 (cit. on p. 36).

[DL13] T. DeMarco, T. Lister. *Peopleware: productive projects and teams*. Addison-Wesley, 2013 (cit. on pp. 20, 53).

[DM05] D. P. Darcy, M. Ma. “Exploring individual characteristics and programming performance: Implications for programmer selection.” In: *System Sciences, 2005. HICSS’05. Proceedings of the 38th Annual Hawaii International Conference on*. IEEE. 2005, 314a–314a (cit. on p. 21).

[DS04] V. Dagiene, J. Skupiene. “Learning by competitions: olympiads in informatics as a tool for training high-grade skills in programming.” In: *Information Technology: Research and Education, 2004. ITRE 2004. 2nd International Conference on*. IEEE. 2004, pp. 79–83 (cit. on p. 19).

[Dum17] Dumitru. *How to Find a Solution - topcoder*. 2017. URL: http://archive.is/aQmhR (cit. on p. 13).

[DWT+10] E. Diener, D. Wirtz, W. Tov, C. Kim-Prieto, D.-w. Choi, S. Oishi, R. Biswas-Diener. “New well-being measures: Short scales to assess flourishing and positive and negative feelings.” In: *Social Indicators Research* 97.2 (2010), pp. 143–156 (cit. on p. 36).

[Ent17] Entrefuse Inc. *Codewars*. 2017. URL: https://www.codewars.com/ (cit. on p. 13).

[ES89] G. E. Evans, M. G. Simkin. “What best predicts computer proficiency?” In: *Communications of the ACM* 32.11 (1989), pp. 1322–1327 (cit. on p. 21).

[FBRP17] D. Ford, T. Barik, L. Rand-Pickett, C. Parnin. “The tech-talk balance: what technical interviewers expect from technical candidates.” In: *Proceedings of the 10th International Workshop on Cooperative and Human Aspects of Software Engineering*. IEEE Press. 2017, pp. 43–48 (cit. on p. 13).

[For10] M. Forišek. “The difficulty of programming contests increases.” In: *International Conference on Informatics in Secondary Schools-Evolution and Perspectives*. Springer. 2010, pp. 72–85 (cit. on pp. 18, 28).

[GF09] G. Gárcia-Mateos, J. L. Fernández-Alemán. “A course on algorithms and data structures using on-line judging.” In: *ACM SIGCSE Bulletin*. Vol. 41. 3. ACM. 2009, pp. 45–49 (cit. on p. 19).

[Gho07] I. Ghory. *Using FizzBuzz to Find Developers who Grok Coding*. 2007. URL: http://archive.is/FGZKG (cit. on p. 13).

[Gin96] D. Ginat. “Efficiency of algorithms for programming beginners.” In: *ACM SIGCSE Bulletin*. Vol. 28. 1. ACM. 1996, pp. 256–260 (cit. on p. 19).

[Gla02] R. L. Glass. *Facts and fallacies of software engineering*. Addison-Wesley Professional, 2002 (cit. on p. 20).

[Goo17] Google Inc. *Google Code Jam*. 2017. URL: http://archive.is/yG5Kb (cit. on p. 23).

[GWA14] D. Graziotin, X. Wang, P. Abrahamsson. “Happy software developers solve problems better: psychological measurements in empirical software engineering.” In: *PeerJ* 2 (2014), e289 (cit. on pp. 20, 52).

[HIKM12] J. Helminen, P. Ihantola, V. Karavirta, L. Malmi. “How do students solve parsons programming problems?: an analysis of interaction traces.” In: *Proceedings of the ninth annual international conference on International computing education research*. ACM. 2012, pp. 119–126 (cit. on p. 20).

[JS99] O. P. John, S. Srivastava. “The Big Five trait taxonomy: History, measurement, and theoretical perspectives.” In: *Handbook of personality: Theory and research* 2.1999 (1999), pp. 102–138 (cit. on p. 36).

[KVC06] G. Kemkes, T. Vasiga, G. Cormack. “Objective scoring for computing competition tasks.” In: *International Conference on Informatics in Secondary Schools-Evolution and Perspectives*. Springer. 2006, pp. 230–241 (cit. on p. 26).

[Lee17] LeetCode. *LeetCode*. 2017. URL: https://leetcode.com/ (cit. on p. 13).

[Lev05] A. Levitin. “Analyze that: puzzles and analysis of algorithms.” In: *ACM SIGCSE Bulletin*. Vol. 37. 1. ACM. 2005, pp. 171–175 (cit. on p. 19).

[LLA01] F. R. Lang, O. Lüdtke, J. B. Asendorpf. “Testgüte und psychometrische Äquivalenz der deutschen Version des Big Five Inventory (BFI) bei jungen, mittelalten und alten Erwachsenen.” In: *Diagnostica* 47.3 (2001), pp. 111–121 (cit. on p. 36).

[LS03] J. P. Leal, F. Silva. “Mooshak: A Web-based multi-site programming contest system.” In: *Software: Practice and Experience* 33.6 (2003), pp. 567–581 (cit. on p. 19).

[McD13] G. L. McDowell. *What are Gayle Laakmann McDowell’s favorite questions to ask in a software engineering interview, and what does she look for in evaluating the candidate’s performance?* 2013. URL: https://archive.is/N31RD (cit. on p. 24).

[McD15] G. L. McDowell. *Cracking the Coding Interview - 189 Programming Questions and Solutions*. 6th. CareerCup, 2015. ISBN: 978-0-984-78285-7 (cit. on pp. 13, 18, 23–26, 28, 34, 54–56).

[McD17] G. L. McDowell. *What is a typical software engineering interview with you like?* 2017. URL: http://archive.is/hR0jL (cit. on p. 24).

[MJ92] R. R. McCrae, O. P. John. “An introduction to the five-factor model and its applications.” In: *Journal of personality* 60.2 (1992), pp. 175–215 (cit. on p. 36).

[MK+12] J. Mongan, N. Kindler, et al. *Programming interviews exposed: secrets to landing your next job*. John Wiley & Sons, 2012 (cit. on pp. 13, 18, 25).

[MN86] D. Myers, L. Null. “Design and implementation of a programming contest for high school students.” In: *ACM SIGCSE Bulletin*. Vol. 18. 1. ACM. 1986, pp. 307–312 (cit. on p. 19).

[Pax07] J. Paxton. “Programming competition problems as a basis for an algorithms and data structures course.” In: *Journal of Computing Sciences in Colleges* 23.2 (2007), pp. 27–32 (cit. on p. 19).

[RCC+16] L. F. Rosales-Castro, L. A. Chaparro-Gutiérrez, A. F. Cruz-Salinas, F. Restrepo-Calle, J. Camargo, F. A. González. “An Interactive Tool to Support Student Assessment in Programming Assignments.” In: *Ibero-American Conference on Artificial Intelligence*. Springer. 2016, pp. 404–414 (cit. on p. 20).

[RHS17] T. Rahm, E. Heise, M. Schuldt. “Measuring the frequency of emotions—validation of the Scale of Positive and Negative Experience (SPANE) in Germany.” In: *PloS one* 12.2 (2017), e0171288 (cit. on p. 36).

[Rig66] J. W. Rigney. *COMPUTER PERSONNEL SELECTION AND CRITERION DEVELOPMENT: 3. THE BASIC PROGRAMMING KNOWLEDGE TEST*. Tech. rep. DTIC Document, 1966 (cit. on p. 21).

[RML08] M. A. Revilla, S. Manzoor, R. Liu. “Competitive learning in informatics: The UVa online judge experience.” In: *Olympiads in Informatics* 2 (2008), pp. 131–148 (cit. on p. 18).

[SEG68] H. Sackman, W. J. Erikson, E. E. Grant. “Exploratory experimental studies comparing online and offline programming performance.” In: *Communications of the ACM* 11.1 (1968), pp. 3–11 (cit. on p. 21).

[Sin07] K. Singh. *Quantitative social research methods*. Sage, 2007 (cit. on p. 27).

[SR06] S. S. Skiena, M. A. Revilla. *Programming challenges: The programming contest training manual*. Springer Science & Business Media, 2006 (cit. on p. 18).

[TB13] N. A. Tonin, J. L. Bez. “Uri online judge: A new interactive learning approach.” In: *Computer Technology and Application* 4.1 (2013) (cit. on pp. 19, 20).

[Urn17] T. Urness. “Using interview questions as short-term programming assignments in CS2.” In: *Journal of Computing Sciences in Colleges* 32.5 (2017), pp. 170–177 (cit. on p. 19).

[VCM+06] T. Vasiga, G. Cormack, I. Munro, G. Kemkes, et al. “Structure, scoring and purpose of computing competitions.” In: *Informatics in Education-An International Journal* Vol 5_1 (2006), pp. 15–36 (cit. on p. 17).

All links were last followed on October 27, 2017.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature