J. Figueiredo, N. Lopes and F. J. García-Peñalvo, "Predicting Student Failure in an Introductory Programming Course with Multiple Back-Propagation," in TEEM'19 Proceedings of the Seventh International Conference on Technological Ecosystems for Enhancing Multiculturality (Leon, Spain, October 16th-18th, 2019), M. Á. Conde-González, F. J. Rodríguez-Sedano, C. Fernández-Llamas and F. J. García-Peñalvo, Eds. ICPS: ACM International Conference Proceedings Series, pp. 44-49, New York, NY, USA: ACM, 2019. doi: 10.1145/3362789.3362925.

Predicting Student Failure in an Introductory Programming Course with Multiple Back-Propagation

José Figueiredo
Research Unit for Inland Development (UDI)
Polytechnic of Guarda
Portugal
jfig@ipg.pt

Noel Lopes
UDI, Polytechnic of Guarda
CISUC, University of Coimbra
Institution/University Name
Portugal
noel@ipg.pt

Francisco José García-Peñalvo
Computer Science Department
Research Institute for Educational Sciences GRIAL
University of Salamanca
Spain
fgarcia@usal.es

ABSTRACT

One of the most challenging tasks in computer science and similar courses consists of both teaching and learning computer programming. Usually this requires a great deal of work, dedication, and motivation from both teachers and students. Accordingly, ever since the first programming languages emerged, the problems inherent to programming teaching and learning have been studied and investigated. The theme is very serious, not only for the important concepts underlying computer science courses but also for reducing the lack of motivation, failure, and abandonment that result from students frustration. Therefore, early identification of potential problems and immediate response is a fundamental aspect to avoid student's failure and reduce dropout rates. In this paper, we propose a machine-learning (neural network) predictive model of student failure based on the student profile, which is built throughout programming classes by continuously monitoring and evaluating student activities. The resulting model allows teachers to early identify students that are more likely to fail, allowing them to devote more time to those students and try novel strategies to improve their programming skills.

KEYWORDS

datasets, neural networks, programming, teaching programming, learning programming, CS0, CS1

1 Introduction

Nowadays there is an on-growing demand for Computer Science (CS) professionals. In Europe alone, according to European Commission, next year there will be a shortage of more than 800,000 CS professionals. To make matters worse, CS courses typically present high-dropout rates, aggravating the shortage of professionals [10, 26]. Accordingly, in recent years, there has been a growing interest in teaching programming concepts to young people. Incorporating CS topics in high-school curriculum's, such as computer programming and robotics, has the benefit of developing students' skills, such as: problem solving, creativity and computational thinking [8, 31]. Recognizing this, many entities and organizations promote countless initiatives to promote computer programming and computational thinking [2, 4, 6, 7, 11, 21, 22].

Programming is a process of transforming a mental plan of cur-rent terms into terms compatible with the computer [13]. When teaching computer programming, the main objective is to empower students with the skills needed to create computer programs that can solve real-world problems. In this context, programming requires quite particular characteristics and skills that students may struggle to obtain, often in a short period. Among these, Jenkins [17] identified the following: the abstract concepts inherent to programming; the competencies and mental

abilities required to decompose and solve problems; the use of specific syntaxes that students are required to memorize; and the semantics and structure new non-natural languages. As such, among the topics CS students are required to learn, computer programming is particularly difficult. This is a well-know fact, among the Computer Science Education (CSE) community [3]. Not only teaching computer programming is a recent area (when compared to other knowledge areas) but it is also a fast-changing one. Knowledge and tools rapidly become obsolete, making it harder to teach and learn even the basics.

Although many studies seek to identify the different causes that lead to a lack of student motivation and high-dropout rates there is no panacea to solve this problem. Nevertheless, to counter students' frustration that may ultimately lead to dropouts, it is important to identify learning problems as soon as possible. Accordingly, in this paper, we propose a Neural Network (NN) predictive model of student failure based on students' profiles collected during the classes. The resulting model allows teachers to early identify students that are more to fail and take corrective actions.

The remainder of the paper is structured as follows. Section 2 describes related work. Section 3 details the algorithm used to create a predictive model for the student's performance. Section 4 explains the methodology used to create the aforementioned model. Section 5 presents and discusses the results obtained and finally, in Section 5 conclusions and future work are addressed.

2 Related work

The first programming courses play an important role in the student success [26], because they can stimulate all passion, beauty, joy, and awe for programming [9]. Unfortunately, the lack of success on the introductory programming courses, can also be a demotivating factor that may ultimately lead to abandonment. Therefore, several works focus on predicting the success of students. Most of these are based on analysis of student interaction with the programming language compiler, such as: investigating novice programming mistakes [1]; exploring compilation behavior [16]; analyzing response to compiler error messages [23]. Other studies focus on comparing and testing traditional predictors of performance and new data-driven predictors [27]. Several works have found that the mathematical ability and exposure to mathematics courses are important predictors of performance on introductory programming courses [25, 28, 30]. Moreover, there are also studies, that link prior programming experience and non-programming computer experience to the student's programming performance [5, 12, 14, 29]. Additionally, the role of cognitive factors, such as problem-solving, abstract reasoning, logical ability, and cognitive style for predicting the programming performance has also been studied [15, 24].

Recently, Machine Learning (ML) methods have also been used for predicting student's performance on introductory programming courses. In [18], support vector machines are used to predict the students' final exam scores based on data automatically collected for instructors. In [26] a prediction model, named PreSS (Predict Student Success), based on ML algorithms is proposed, in order to predict the students' success on introductory programming courses. Moreover, the study, analyses and compares the performance of several ML methods.

3 Multiple Back-Propagation with a Neural Selective Input Model

Multiple Back-Propagation (MBP) is an algorithm, proposed in [20] for training Multiple Feed-Forward (MFF) networks – a special type of NN that combines a main network and space network as depicted in Figure 1. The main network contains selective activation neurons, whose contribution to the NN output depends on the stimulus presented to the network. Each selective activation neuron, k, possesses an importance factor m_k^p, that defines its relevance according to the pattern (sample), p, presented to the network. Accordingly, the output of these neurons, y_k^p, is given by (1):

$$y_k^p = m_k^p \mathcal{F}_k(a_k^p) = m_k^p \mathcal{F}_k(\sum_j \quad w_{jk} y_j^p + \theta_k), \qquad (1)$$

where $\mathcal{F}_k$ is the neuron activation function, a_k^p its activation, θ_k the bias and w_{jk} the weight of the connection between neuron j and neuron k. The farther from zero m_k^p is the more important the neuron contribution becomes. On the other hand, a value of zero means the neuron is completely irrelevant for the network output and one can interpret such a value as if the neuron is not present in the network.

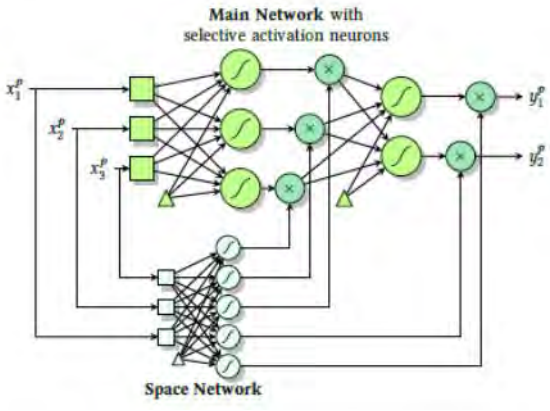

Figure 1: Example of a MFF NN with 3 inputs and 2 outputs. Squares represent inputs, darker circles (with the symbol×) multipliers, lighter circles neurons, and triangles bias [20].

Both the main network and the space network, responsible for determining the importance factors of the selective neurons, are trained together as a whole. To that end, MBP uses the same rule for updating the weights as the Back-Propagation (BP) algorithm, i.e. the main network weights are adjusted by (2):

$$\Delta_p w_{jk} = \gamma \delta_k^p y_j^p + \alpha \Delta_l w_{jk} \quad (2)$$

where γ is the learning rate, δ_k^p the local gradient of neuron k, $\Delta_l w_{jk}$ the weight change for the last pattern l, α the momentum term and the local gradient for the output, o, and hidden, h, neurons, are given respectively by (3) and (4):

$$\delta_o^p = (d_o^p - y_o^p) m_o^p \mathcal{F}_o{}'(a_o^p), \quad (3)$$

$$\delta_h^p = m_h^p \mathcal{F}_h{}'(a_h^p) \sum_{o=1}^{N_o} \delta_o^p w_{ho}. \quad (4)$$

To deal with missing data a Neural Selective Input Model (NSIM) was proposed in [19], such that the act of obtaining the value of x_j^p, represented by a random variable, r_j^p ~ Be(qj) (with Bernoulli distribution), is taken into consideration. To that end, the values x_j^p are transformed by (5):

$$\tilde{x}_j^p = r_j^p \mathcal{F}_k(w_{jk} x_j^p + \theta_k). \quad (5)$$

using a neuron, k, with selective activation (named selective input), containing a single input, x_j^p, and an importance factor m_k^p set to r_j^p, as depicted in Figure 2.

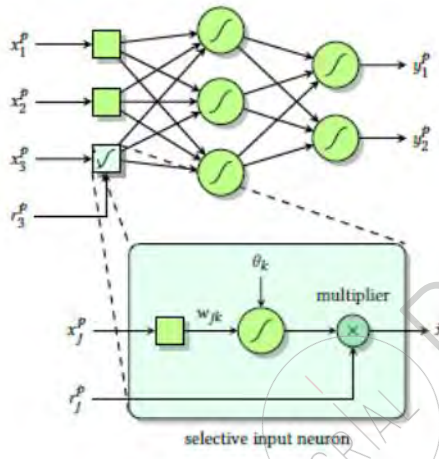

Figure 2: Network with a selective input (k = 3).

4 Methodology work

In order to build a predictive model for determining which students are more likely to fail, a profile for each student was built as described in Section 4.1. The collected data was then divided into training and test datasets. This process is described in Section 4.2. The training dataset was then used to build the model while the test dataset was used to validate it. Section 4.3 presents the metrics used to assert the quality of the models.

4.1 Building the student profiles

The idea of building a profile with the student's programming competencies is based on the same concept of current video games such as FIFA 19 or Assassin's Creed. The characters are invited to build and improve their characteristics and skills in specific areas to complete their tasks or change their level. For example, a FIFA player may train a penalty, dribble, free kicks and corner kicks practice and other actions to improve his/her abilities during the game. Likewise, we want each student to be able to improve and deepen their programming skills by performing a set of appropriate and worked exercises for each student and situation.

The profiles, build throughout the programming classes, by continuously monitoring and evaluating student activities will then be used to build a predictive model able to determine which students require more help in order to overcome their difficulties and achieve the necessary set of programming competences (see Figure 3).

Table 1 describes the 17 variables (attributes) collected in order to build each student profile.

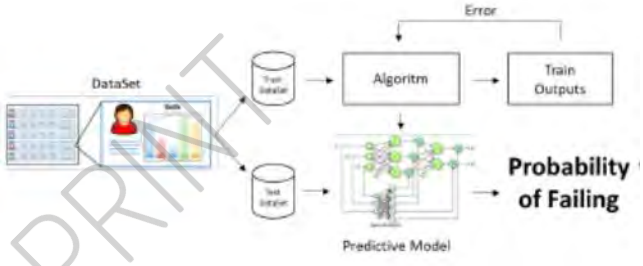

Figure 3: Building the model to predict the probability of retention of students in an introductory programming course.

Table 1: Student profile attributes collected using information provided by the students and teacher.

Attribute	Description
participation	Participation in classes
curiosity	Curiosity and initiative in doing new activities
perfectionist	Try to write programs in the best possible way
passion	The passion, joy, and admiration, demonstrated by the student, towards programming
spatial_ability	Score in activities related to the detection of cognitive reasoning abilities and spatial visualization (*e.g.* punched holes)
follow_instructions	Capacity of give and following instructions
basics	Capacity to define the structure of a program, identifiers, data type, read and write data
if	Capacity of using an `if` conditional structure
switch	Capacity of using a `switch` structure
for	Capacity of using the a `for` iterative structure
do-while	Capacity of using a `do-while` structure
while	Capacity of using the a `while` structure
activity_01	Activities to evaluate the students knowledge using different `C` programming structures
activity_02	
activity_03	
activity_04	
activity_05	

4.2 Data analysis and pre-processing

This study involved a group of 85 students of an introductory programming course (Introduction to Programming), lectured to the first year, first semester, students of the CS course at the Polytechnic of Guarda (IPG), Portugal – an institution of higher education located in the interior of the country. In this course the C language is used to teach the basic programming concepts.

In our opinion, our study group has very special characteristics that might affect the learning process:

• Usually, the IPG Computer Science course, is not the first choice of students. Naturally, this affects negatively student's motivation and engagement.

• In recent years, the score needed to enroll the IPG Computer Science course is typically low. Thus, most students entering this course are below average.

• Most students reveal general difficulties in the area of CS.

• Typically, students were never exposed to programming concepts nor had the opportunity to practice computational thinking activities.

Since students don't always attend classes, the compiled dataset contains Missing Values (MV). On average 13.01 ± 2.70 attributes were collected for each student and we were able to collect all the attributes only for 7 students. The resulting dataset, containing 23.46% of MV was randomly divided into a training set containing 40% of the samples (students) and a test set, containing 60 of the samples (observe Figure 3). Accordingly, the training dataset encompasses 34 samples and the test dataset encompasses the remaining 51 samples. Moreover, the training dataset contains 24.91% of MV while the test dataset contains 22.49% of MV. On average, each student on the training dataset has 12.76 ± 3.03 attributes and each student on the test dataset 13.18 ± 2.47 attributes.

Table 2 presents the percentage of MV per attribute for each dataset. Moreover, Table 3 presents the number of students that approved/failed in each dataset.

Table 2: Percentage of MV in each dataset.

Attribute	Dataset		
	Original	Train	Test
participation	0.00%	0.00%	0.00%
curiosity	0.00%	0.00%	0.00%
perfectionist	0.00%	0.00%	0.00%
passion	61.18%	61.76%	60.78%
spatial_ability	48.24%	52.94%	45.10%
follow_instructions	61.18%	58.82%	62.75%
basics	0.00%	0.00%	0.00%
if	0.00%	0.00%	0.00%
switch	0.00%	0.00%	0.00%
for	0.00%	0.00%	0.00%
do-while	0.00%	0.00%	0.00%
while	0.00%	0.00%	0.00%
activity_01	50.59%	52.94%	49.02%
activity_02	29.41%	41.18%	21.57%
activity_03	41.18%	41.18%	41.18%
activity_04	44.71%	50.00%	41.18%
activity_05	62.35%	64.71%	60.78%

Table 3: Number of students that approved in the first exam of introduction to Programming course.

	Approved	Failed	Total
Train dataset	4	30	34
Test dataset	8	43	51
Original dataset	12	73	85

4.3 Training and evaluating the models

In order to build a NN predictive model for student failure, Multiple Back-Propagation software, available at http://mbp.sourceforge.net/ was used. Several networks were trained using the train dataset while varying the number of neurons in the hidden layer.

For evaluating the performance of each network (model) and asserting its quality, we use several metrics, based on the confusion matrix, which contains the number of correctly and incorrectly classified examples for each class (Failed/Approved), in the form of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN). Figure 4 presents the confusion matrix for our problem. The accuracy given by (6):

$$accuracy = \frac{TP+TN}{TP+FP+FN+TN} . \quad (6)$$

The accuracy represents the proportion of students that are correctly classified. Although this metric gives an overall estimate of our model performance, in our case, it can be misleading, since there is a big discrepancy between the number of samples of each class (see Table 3). Therefore, we also use other metrics, such as the precision and recall (sensitivity), respectively given by (7) and (8):

$$precision = \frac{TP}{TP+FP}, \quad (7)$$

$$recall = sensitivity = \frac{TP}{TP+FN}. \quad (8)$$

A model presenting a high precision rate is rarely wrong when it predicts that a student will fail the course. On the other hand, a classifier exhibiting a high recall rate rarely mis-classifies a student that will fail. Usually there is a trade-off between the precision and the recall, and generally it is important to balance and maximize both. This can be accomplished by using the F-score (9):

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall}. \quad (9)$$

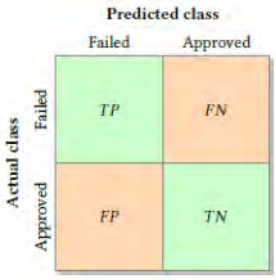

Figure 4: Confusion matrix of the model to predict which students will fail the Introduction to Programming course.

5 Results and Discussion

In order to build a NN predictive model for student failure, Multiple Back-Propagation software, available at http://mbp.sourceforge.net/ was used. Several networks were trained, varying the number of hidden neurons in the main network. The best network had a single neuron with selective activation. Figure 5 presents the confusion matrix of the resulting NN model in the test dataset. Moreover, Table 4 presents the performance of the model in the test data.

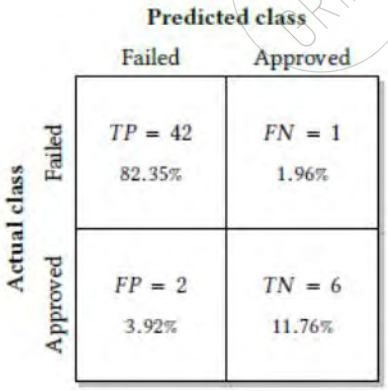

Figure 5: Confusion matrix of the NN model for predicting student success in the dataset.

Table 4: Performance of the NN predictive model for student failure.

Accuracy	94.12%
Precision	95.45%
Recall	97.67%
F_1	96.55%

The resulting NN model presents high-accuracy. Only three students are misclassified. However, as depicted in Figure 5, from these, only one is incorrectly classified as approved when it should be classified as failed (FN). Ideally this value would be zero, since these students will not be given special attention, because the model predicts they will succeed and unfortunately, they will fail. Nevertheless, the proposed model allows the teacher to accurately determine which students require additional attention and intervene early on in order to reduce attrition rates.

6 Conclusion

In this paper we presented a Neural Network predictive model for student failure based on the student profile, which is built throughout computer programming classes by continuously monitoring and evaluating student activities. The resulting model allows teachers to early identify students that are more likely to fail, allowing them to devote more time to those students and try novel strategies to improve their programming skills. Future work will focus on more gathering additional information to explore the possibility of creating models for determining the students' performance on specific contents (e.g. defining variables, using iterative or conditional structures).

REFERENCES

[1] Altadmri, A. and Brown, N.C.C. 2015. 37 Million Compilations: Investigating Novice Programming Mistakes in Large-Scale Student Data. *Sigcse '15*. (2015). DOI:https://doi.org/10.1145/2676723.2677258.

[2] Basogain, X. et al. 2017. Computational Thinking in pre-university Blended Learning classrooms. *Computers in Human Behavior*. (May 2017). DOI:https://doi.org/10.1016/j.chb.2017.04.058.

[3] Bergin, S. and Reilly, R. 2005. Programming: Factors that Influence Success. *SIGCSE '05: Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education* (St. Louis, Missouri, United States, 2005), 411–415.

[4] Cole, E. 2015. On Pre-requisite Skills for Universal Computational Thinking Education. (2015), 253–254. DOI:https://doi.org/10.1145/2787622.2787737.

[5] Evans, G.E. and Simkin, M.G. 1989. What best predicts computer proficiency? *Communications of the ACM*. 32, 11 (1989), 1322–1327. DOI:https://doi.org/10.1145/68814.68817.

[6] Garcia-Peñalvo, F.J. 2016. What Computational Thinking Is. *Journal of Information Technology Research*. 9(3), v–vi, October (2016).

[7] García-Peñalvo, F.J. et al. 2016. Evaluation Of Existing Resources (Study/Analysis). (Jan. 2016). DOI:https://doi.org/10.5281/ZENODO.163112.

[8] García-Peñalvo, F.J. and Mendes, A.J. 2017. Exploring the computational thinking effects in pre-university education. *Computers in Human Behavior*. (Dec. 2017). DOI:https://doi.org/10.1016/j.chb.2017.12.005.

[9] Garcia, D.D. et al. 2016. Rediscovering the passion, beauty, joy, and awe: {Making} computing fun again, part 7. *Proceedings of the 47th ACM Technical Symposium on Computer Science Education (SIGCSE '16)*. (2016), 273–274. DOI:https://doi.org/10.1145/2538862.2538874.

[10] Giannakos, M.N. et al. 2017. Understanding student retention in computer science education: The role of environment, gains, barriers

and usefulness. *Education and Information Technologies*. 22, 5 (2017), 2365–2382. DOI:https://doi.org/10.1007/s10639-016-9538-1.

[11] Grover, S. and Pea, R. 2013. Computational Thinking in K-12: A Review of the State of the Field. *Educational Researcher*. 42, 1 (2013). DOI:https://doi.org/10.3102/0013189X12463051.

[12] Hagan, D. and Markham, S. 2004. Does it help to have some programming experience before beginning a computing degree program? *ACM SIGCSE Bulletin*. 32, 3 (2004), 25–28. DOI:https://doi.org/10.1145/353519.343063.

[13] Hoc, J.-M. and Nguyen-Xuan, A. 1990. Language Semantics, Mental Models and Analogy. *J.-M. Hoc, T. R. G. Green, R. Samurçay, & D. J. Gilmore (Eds.), Psychology of Programming*. (1990), 139–156.

[14] Holden, E. and Weeden, E. 2004. The impact of prior experience in an information technology programming course sequence. (2004), 41. DOI:https://doi.org/10.1145/947121.947131.

[15] Hostetler, T.R. 1983. Predicting Student Success in an Introductory Programming Course. *SIGCSE Bull.* 15, 3 (1983), 40–43. DOI:https://doi.org/10.1145/382188.382571.

[16] Jadud, M.C. 2006. Methods and tools for exploring novice compilation behaviour. *Proceedings of the Third International Workshop on Computing Education Research*. Figure 1 (2006), 73–84. DOI:https://doi.org/10.1145/1151588.1151600.

[17] Jenkins, T. 2002. On the Difficulty of Learning to Program. *Language*. 4, (2002), 53–58. DOI:https://doi.org/10.1109/ISIT.2013.6620675.

[18] Liao, S.N. et al. 2019. A Robust Machine Learning Technique to Predict Low-performing Students. *ACM Transactions on Computing Education*. 19, 3 (2019), 1–19. DOI:https://doi.org/10.1145/3277569.

[19] Lopes, N. and Ribeiro, B. 2012. Handling Missing Values via a Neural Selective Input Model. *Neural Network World*. 22, (2012), 357–370. DOI:https://doi.org/10.14311/NNW.2012.22.021.

[20] Lopes, N. and Ribeiro, B. 2001. Hybrid learning in a multi-neural network architecture. *INNS-IEEE International Joint Conference on Neural Networks (IJCNN 2001)* (2001), Vol-4. 2788-2793.

[21] Lye, S.Y. and Koh, J.H.L. 2014. Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*. 41, (2014), 51–61. DOI:https://doi.org/10.1016/j.chb.2014.09.012.

[22] Mason, D. et al. 2016. Computational Thinking as a Liberal Study. *Proceedings of the 47th ACM Technical Symposium on Computer Science Education (SIGCSE '16)*. (2016), 24–29. DOI:https://doi.org/10.1145/2839509.2844655.

[23] Munson, J.P. and Schilling, E.A. 2016. Analyzing Novice Programmers' Response to Compiler Error Messages. *J. Comput. Sci. Coll.* 31, 3 (2016), 53–61.

[24] Nowaczyk, R. 2019. Cognitive skills needed in computer programming paper presented at the annual meeting of the southeastern psychological association (march. (2019).

[25] Patil, S.P. and Goje, A.C. 2009. The effect of developments in student attributes on success in programming of management students. *2009 International Conference on Education Technology and Computer, ICETC 2009*. (2009), 191–193. DOI:https://doi.org/10.1109/ICETC.2009.35.

[26] Quille, K. and Bergin, S. 2019. CS1: how will they do? How can we help? A decade of research and practice. *Computer Science Education*. 29, 2–3 (2019), 254–282. DOI:https://doi.org/10.1080/08993408.2019.1612679.

[27] Watson, C. et al. 2014. No tests required: comparing traditional and dynamic predictors of programming success. *Proceedings of the 45th ACM technical symposium on Computer science education - SIGCSE '14*. 44, July (2014), 469–474. DOI:https://doi.org/10.1145/2538862.2538930.

[28] Werth, L.H. 1986. Predicting Student Performance in a Beginning Computer Science Class. *SIGCSE Bull.* 18, 1 (1986), 138–143. DOI:https://doi.org/10.1145/953055.5701.

[29] Wiedenbeck, S. et al. 2004. Factors affecting course outcomes in introductory programming. *16th Workshop of the Psychology of Programming Interest Group*. April (2004), 97–110. DOI:https://doi.org/10.1.1.103.9447.

[30] Wilson, B.C. and Shrock, S. 2004. Contributing to success in an introductory computer science course. *ACM SIGCSE Bulletin*. 33, 1 (2004), 184–188. DOI:https://doi.org/10.1145/366413.364581.

[31] Wing, J.M. 2006. Computational thinking. Communications of the ACM. 49, 3 (Mar. 2006), 33. DOI:https://doi.org/10.1145/1118178.1118215.