MODESTUM
OPEN ACCESS

EURASIA Journal of Mathematics, Science and Technology Education, 2018, 14(10), em1591
ISSN:1305-8223 (online)
Research Paper https://doi.org/10.29333/ejmste/93190

Exploring Bimodality in Introductory Computer Science Performance Distributions

Ram B Basnet [1], Lori K Payne [1], Tenzin Doleck [2*], David John Lemay [2], Paul Bazelais [2]
[1] Colorado Mesa University, Grand Junction, Colorado, USA
[2] McGill University, Montreal, Quebec, CANADA

Received 7 April 2018 ▪ Revised 19 June 2018 ▪ Accepted 21 June 2018

ABSTRACT
This study examines student performance distributions evidence bimodality, or whether there are two distinct populations in three introductory computer science courses grades at a four-year southwestern university in the United States for the period 2014-2017. Results suggest that computer science course grades are not bimodal. These findings counter the double hump assertion and suggest that proper course sequencing can address the needs of students with varying levels of prior knowledge and obviate the double-hump phenomenon. Studying performance helps to improve delivery of introductory computer science courses by ensuring that courses are aligned with student needs and address preconceptions and prior knowledge and experience.

Keywords: computer science performance, coding, programming, double hump, grade distribution, bimodal distribution, unimodal distribution

BACKGROUND

In recent years, there has been a resurgence of interest in the practice of coding (Kafai & Burke, 2013), with many pushing for making it a core competency for students (Lye & Koh, 2014). There are inherent challenges in learning to code evidenced by high failure and dropout rates in programming courses (Ma, Ferguson, Roper, & Wood, 2011; (Qian & Lehman, 2017; Robins, 2010). These crucial issues go to the core of teaching coding (Robins, Rountree, & Rountree, 2003; Watson & Li, 2014) and demand our attention given the growing need for coders across a broad range of careers as "seven million job openings in 2015 were in occupations which value coding skills" (Burning Glass, 2016, p. 3; Dishman, 2016; Thompson, 2018). The computer science education community has recognized the importance of better understanding students' performance in computer science courses for improving student outcomes (Alturki, 2016; Ott, Robins, Haden, & Shephard, 2015; Zingaro, 2015), and many have noted a bimodal distribution of grades in computer science (Corney, 2009; Dehnadi & Bornat, 2006; Robins, 2010). These researchers suggest that there are two distinct groups of computer science students, one stronger and one weaker, that can even be observed in distributions of learning outcomes in introductory computer science courses.

Over a decade ago, Dehnadi and Bornat (2006) advanced the notion of double hump, positing that "there at least two populations in initial programming courses" (Dehnadi & Bornat, 2006, p.16). Or as Robins (2010) put it, "the typical introductory programming (CS1) course has higher than usual rates of both failing and high grades, creating a characteristic bimodal grade distribution" (p.37). However, supporting evidence remains inconclusive and this contention remains rather controversial to date as it implies that students either get it or they don't. As Dehnadi and Bornat (2006) reasoned, "programming teaching is useless for those who are bound to fail and pointless for those who are certain to succeed" (p. 1). In the computer science education literature various explanations for the purported bimodal grade distribution have been offered, including: the geek gene hypothesis, prior knowledge (students with and without experience), stumbling point hypothesis, threshold concepts, learning edge momentum effect, and poor assessment strategies (Ahadi & Lister, 2013; Lister, 2010; Patitsas, Berlin, Craig,

✉ rbasnet@coloradomesa.edu ✉ lpayne@coloradomesa.edu
✉ tenzin.doleck@mail.mcgill.ca **(*Correspondence)** ✉ david.lemay@mail.mcgill.ca
✉ paul.bazelais@mail.mcgill.ca

Contribution of this paper to the literature

- Performance distributions from three introductory computer science courses reveal unimodal distributions contrary to the conjecture of bimodality of student performance distribution.
- High skewness and kurtosis of performance distributions are characteristic of criterion-based rather than norm-referenced assessments.
- Faculty course sequence planning can address the needs of two distinct student populations by instituting an optional course for those with no prior programming experience.

& Easterbrook, 2016; Robins, 2010). Roughly speaking, these hypotheses are either related to student or course characteristics. Those related to student characteristics state that some students have an ability for programming, or that students' prior experience and knowledge favorably predispose them to programming. Related to course characteristics, researchers have suggested that some concepts have specific antecedents to understanding, and lack of those prerequisite concepts may act as threshold or stumbling point to further progress, disadvantaging those that do not possess the requisite knowledge or skill to advance with the rest of the group.

An assertion such as the one by Dehnadi and Bornat's (2006) was bound to prompt a reaction: the work piqued the interest of computer science education researchers, and set in motion two general channels of studies that have tried to both replicate and explicate the idea of double hump in computer science. The first stream of work involves using the aptitude test proposed by Dehnadi and Bornats (2006) to illustrate the double hump phenomenon. Much of the research has failed to find support for the aptitude test (e.g., Caspersen, Larsen, & Bennedsen, 2007; Lung, Aranda, Easterbrook, & Wilson, 2008; França, da Cunha & da Silva, 2010; Wray, 2007).

The second stream of work has focused on examining performance data to ascertain the existence of the double hump. The research reported in this article falls in this second stream of work. Several ensuing contributions are worth highlighting. Robins (2010) explored simulated grade distribution by running a simulation study to provide an account (learning edge momentum effect) for the bimodal grade distributions. Höök and Eckerdal (2015) provided evidence for bimodal distribution of the final grades in an introductory programming course, but noted that it "depends on the correction procedure rather than the distribution of the results of the exam" (p. 79).

Ahadi and Lister (2013), who examined four tests from an introductory programming course, highlighted the complexity of accounting for the bimodal grade distribution and noted that: "advocates of the various hypotheses—Geek Genes, Prior Knowledge, Stumbling Points and Learning Edge Momentum —can all find support for their respective hypotheses, in aspects of the data in this paper" (p. 126). Using 778 distributions of final course grades, Patitsas, Berlin, Craig, and Easterbrook (2016) found that only 5.8% of the distributions passed tests of multimodality. Moreover it is common to find general statements and anecdotes in the literature about the bimodality phenomenon without relevant evidence. For example, Corney (2009) noted that: "Faculty data typically has shown a bimodal distribution of results for students undertaking introductory programming courses with a high proportion of students receiving a high mark and a high proportion of students receiving a low or failing mark" (p. 1). The anecdotal statement is accepted on face value and no further supporting evidence is provided. Furthermore, Patitsas, Berlin, Craig, and Easterbrook (2016) noted that prior work on the bimodality distribution phenomenon in CS performance has generally lacked robust statistical testing, with many resorting to visual inspections of distributions to assess bimodality (Lister, 2010). Hence, there is a need to further understand the topic by conducting robust statistical testing on actual performance data from computer science courses. Finally, a recent development comes in the form of a retraction notice from one of the co-authors of the original article, who acknowledged errors in the original paper (Bornat, 2014); nevertheless, the debate on the topic continues.

The present research conducted robust statistical tests on student performance data from three introductory courses in computer science in an effort to enrich our understanding of the purportedly bimodal distribution in achievement scores in the context of computer science.

Present Study

Most of the existing evidence for and against the double hump comes from the test for programming aptitude proposed by Dehnadi and Bornat (2006). A smaller set of studies uses actual performance data such as final grades from computer science courses (Patitsas, Berlin, Craig, & Easterbrook, 2016). Computer science course performance data provide a real-life context for examining the double hump phenomenon.

To gain better insight into the phenomenon, the present retrospective exploratory study drew on data overall course grade from three computer science courses (CSCI 110, CSCI 111, and CSCI 112) over a three-year period from 2014-2017 at a southwestern university in the United States to answer the following research question:

Does the phenomenon of double hump exist in computer science course grades?

To address the overarching research question, we analyzed the data in the spirit of Patitsas et al. (2016) and provide the following statistics: kurtosis, skewness, Shapiro-Wilk Test of Normality, and Hartigan's dip test statistic (Hartigan & Hartigan, 1985) for unimodality/multimodality. The values for kurtosis, skewness, Shapiro-Wilk Test of Normality were calculated using SPSS. The dip test statistic was calculated using the *R* package dip test.

CONTEXT: DESCRIPTION OF COURSES

CSCI 110: Beginning Programming: Python

CSCI 110 (3 credit hours) is an introduction to programming course with a prerequisite of MATH 110 (College Algebra) or Math 119 (Pre-Calculus). Students from various disciplines take this course as an introduction to the field of Computer Science. Additionally, computer science students with no prior programming experience or those who do not meet the MATH 110 prerequisite for CSCI 111 course typically take this course. The course covers some basics of Linux and students learn to code in the Linux environment using Python as the programming language to learn basic programming concepts such as data types, variables, functions, automated testing, I/O, loops, conditionals, and Python built-in libraries and data types. CSCI 110 is not a required or core-course for Associate or Bachelor's Degrees in Computer Science. One section of this course is offered every semester with a maximum student enrollment of 30. There's an optional lab (1 credit hour) component that we excluded from this study. Weekly homework is assigned to assess and enforce the concepts covered. Some professors assign problems from open.kattis.com and submit solutions to Kattis online judge for testing. This practice has been studied and it has been found that most students are very engaged with Kattis and they continue to use Kattis beyond the classroom (Basnet, Doleck, Lemay, & Bazelais, 2018). In some cases, short quizzes and tests are also given to further assess students' understanding of the concepts and retention of the materials. Typically, students are assigned an individual final project towards the end of the semester.

CSCI 111: Foundation of Computer Science

CSCI 111 (4 credit hours) is the foundational course that covers problem solving techniques emphasizing modularity, abstraction, analysis, and correctness of algorithm design. Using C/C++ language as a tool, topics covered include data types, control structures, I/O, functions, struct, and some object-oriented concepts. MATH 113 (College Algebra) or CSCI 110 is a prerequisite for this course. CSCI 111 is a required core-course for Computer Science majors. Other disciplines, such as Mathematics and Engineering, may also require their students to take this course. Depending on the demand for the semester, 2 to 3 sections (max 35 students) of CSCI 111 are offered every semester. About 5-7 quizzes and 6-8 homework assignments are given throughout the semester to assess students' learning. Either a final project or a comprehensive exam is assigned towards the end of the semester. Assignments typically cover problems that emphasize the concepts covered during the course. Similar to CSCI 110, some professors assign problems from open.kattis.com in this course as well.

CSCI 112: Data Structures

CSCI 112 (4 credit hours) is the continuation of CSCI 111. The course emphasizes algorithm design and analysis, procedural abstraction, data abstraction, and quality programming style. Topics covered include distinction between dynamic and static variables; run-time exception handling; automated testing; various implementations of elementary stacks; queues, trees and linked lists; comparison of recursive and iterative algorithms; program correctness; and, hierarchical design principles. Depending on student enrollment, 1 to 2 sections (max 35 students) are offered every semester. About 6-8 homework assignments and 3-5 tests are given throughout the semester to emphasize students' learning and assessment. Some professors assign problems from Kattis in this course as well.

PARTICIPANT PROFILE

The responsible Institutional Review Board approved the current study. Anonymized data were obtained from the registrars after obtaining ethical approval for the study. The summary of the data (age, gender, ethnicity, and course load) is provided below.

Table 1. Age Distribution of Students

	Course Taken						
Age Group	**CSCI 110**		**CSCI 111**		**CSCI 112**		**Total**
24 or younger	63	75%	353	83%	176	80%	592
25 or older	21	25%	72	17%	43	20%	136
	84		425		219		728

Table 2. Gender of Students

	Course Taken						
Gender	**CSCI 110**		**CSCI 111**		**CSCI 112**		**Total**
Male	69	82%	347	82%	190	87%	606
Female	15	18%	78	18%	29	13%	122
	84		425		219		728

Table 3. Ethnicity of Students

	Course Taken						
Ethnicity	**CSCI 110**		**CSCI 111**		**CSCI 112**		**Total**
Asian	3	4%	6	1%	7	3%	16
Pacific Islander	0	0%	3	1%	1	0%	4
African American	0	0%	7	2%	3	1%	10
Hispanic	14	17%	62	15%	28	13%	104
Native American	2	2%	0	0%	0	0%	2
Multi-Racial	2	2%	13	3%	6	3%	21
White	55	65%	308	72%	160	73%	523
Non-Resident Alien	7	8%	15	4%	7	3%	29
Unknown	1	1%	11	3%	7	3%	19
	84		425		219		728

Table 4. Course Load of Students

	Course Taken						
Student Load	**CSCI 110**		**CSCI 111**		**CSCI 112**		**Total**
Fulltime (12+ hrs)	71	85%	385	91%	202	92%	658
Parttime (<12 hrs)	13	15%	40	9%	17	8%	70
	84		425		219		728

ANALYSIS

Skewness and kurtosis statistics and frequency distributions, reported below, were inspected for evidence of bimodality. Contrary to the double hump assertion, the course grade distributions for the computer science courses in the present study were largely unimodal. In the dataset presented above, 90% of the cases displayed unimodal grade distributions. The findings are similar to Patitsas et al. (2016), who found that, for the final computer science grades, about 5.8% of cases were bimodal distributions. Additionally, we found that a large proportion of the grade distributions were normal, also in line with the findings of Patitsas et al. (2016).

Table 5. Statistical Results For CSCI 110

				Hartigan's Dip Test	
Semester	**Kurtosis**	**Skewness**	**Shapiro-Wilk Normality Test**	**D**	***p*-value**
Fall 2014	2.399	1.228	Normal	0.1	0.9029
Spring 2015	0.868	0.552	Normal	0.1	0.9029
Fall 2015	-2.407	-0.166	Normal	0.2	$< 2.2e^{-16}$
Spring 2016	4.833	2.185	Not Normal	0.1	0.9029
Fall 2016	2.000	-1.145	Normal	0.1	0.9029
Spring 2017	4.028	1.981	Not Normal	0.1	0.9029

Table 6. Statistical Results For CSCI 111

				Hartigan's Dip Test	
Semester	**Kurtosis**	**Skewness**	**Shapiro-Wilk Normality Test**	**D**	***p*-value**
Spring 2014	-0.598	-0.183	Normal	0.1200	0.5122
Fall 2014	1.055	0.513	Normal	0.1000	0.9029
Spring 2015	2.738	1.628	Normal	0.1200	0.5122
Fall 2015	2.854	1.498	Normal	0.1000	0.9029
Spring 2016	-1.137	0.383	Normal	0.1333	0.4077
Fall 2016	0.738	0.259	Normal	0.1000	0.9029
Spring 2017	2.283	-1.314	Normal	0.1000	0.9029

Table 7. Statistical Results For CSCI 112

				Hartigan's Dip Test	
Semester	**Kurtosis**	**Skewness**	**Shapiro-Wilk Normality Test**	**D**	***p*-value**
Spring 2014	-1.418	-0.117	Normal	0.1333	0.4077
Fall 2014	2.399	-1.228	Normal	0.1000	0.9029
Spring 2015	-0.846	0.364	Normal	0.1667	0.1645
Fall 2015	-2.763	-0.134	Normal	0.1571	0.2286
Spring 2016	3.040	1.702	Normal	0.1000	0.9029
Fall 2016	-0.612	-0.512	Normal	0.2000	$< 2.2e^{-16}$
Spring 2017	1.282	0.486	Normal	0.1000	0.9029

DISCUSSION

In the present study, we found no support for the bimodal distribution. Indeed, what is striking is the strength of the unimodality in the distributions. The few non-normal distributions have elevated values of kurtosis and were positively skewed, suggesting a tight distribution around a unique mean. Non-normality does not imply multimodality, and indeed, these distributions do not significantly depart from unimodality on Hartigan's dip test. The elevated values of kurtosis suggest low variance in scores and the prevalence of positive skewness in CSCI 110 and CSCI 111 suggests more students are meeting or exceeding course expectations in the first two courses. Whereas performance in CSCI 112 exhibits a trend toward less kurtosis and skewness, and consequently, more normal distributions.

Rather than supporting the bimodal hypothesis, these data describe a situation where performance in CSCI 110 and CSCI 111 does not appear sufficiently discriminating between weaker and stronger students, hence the prevalence of positive skewness and high-levels of kurtosis in grade distributions. Positive skewness in performance is representative of situations where more grades are clustered above the mean than below. This is common in criterion-referenced situations where course performance is a function of completing assignments and test items are chosen for content-coverage but not for discriminating between different performance levels (Brown, 2014; Dunn, Perry, & Morgan, 2002; Sadler, 2005). We observe more normality in the later CSCI 112 as this course may offer more conceptual difficulty than its two prerequisite courses and may be naturally more discriminating.

Such results may not be surprising to faculty policymakers as the presence of the non-compulsory introductory course CSCI 110 suggests that the course was created to address the gap in prior knowledge and skill possessed by the highly varied student population that subscribe to these courses. Students in Mathematics and Science are expected to already possess the minimum competence addressed by the first introductory course and are not required to take the first course on fundamentals. Indeed, this first non-compulsory course can be seen as implicitly addressing the needs of two different groups of students taking introductory computer science courses. While it would be erroneous to claim the bimodal distribution based on the implementation of a preliminary course on computer science fundamentals, it does admit the recognition of different groups of students entering with different needs. We note that comparatively few students (n = 84) take the non-compulsory CSCI 110 while a majority takes CSCI 111 (n =425), but only half as many continue to CSCI 112 (n = 219). Thus, CSCI 111 serves as a threshold course, nearly half of the students that take CSCI 111 do not continue to CSCI 1112. Students choose computer science for myriad reasons, and the course structure reflects that reality. Many students that take an introductory course do not go on to major in that discipline. Whereas criterion-referenced examinations are increasingly common with the rise in competency-based evaluation frameworks, using norm-referenced assessments in addition to criterion-referenced assessments could serve as motivating factor, as the prevalence of competitive coding platforms suggests. Indeed, Kattis includes such a gamified aspect to its coding challenges too (Basnet et al., 2018). Studying performance helps to improve delivery of introductory computer science courses by ensuring that courses are aligned with student needs and address preconceptions and prior knowledge and experience.

The present analysis supports the view that introductory computer science does provide some challenges as students exhibit a range of ability and interests upon entering the introductory computer science course sequence. However, the high-levels of kurtosis and skewness suggest that a more tightly coupled and carefully planned course sequence may help students progress by guiding them to understanding the more difficult concepts through targeted norm-referenced assessment. It is vital that all students have the prerequisite conceptual understanding as well as the basic knowledge and skill covered in a first computer science fundamentals course. This may be best achieved by instituting a norm-referenced model with software like Kattis (Basnet et al., 2018) to evaluate understanding in addition to criterion-based performance measures.

LIMITATIONS AND FUTURE DIRECTIONS

Changes occurred over the time of the study which could have confounded the results. MATH 113 was changed from a prerequisite to a co-requisite for CSCI 111 for the last several years, which might have allowed more weakly prepared students into the class. However, that would be expected to increase the number of students in a lower hump, but it did not. Second, an influx of students who had taken an AP Computer Science course occurred during the study (as a result of grant money encouraging that offering). That change could have encouraged more low grades for CSCI 112, while encouraging higher grades in CSCI 111 and CSCI 110. While those do not appear to have happened, they were not tracked for this study.

Also, prior to the collection of this data began, major efforts were incorporated in the computer science courses to increase retention for the major. These changes included using peer tutoring in labs outside of class, including one or two lab aides from the more advanced computer science majors to help students on lab days—increasing positive interactions and satisfaction for students, and group advising to help students choose the best courses for their background and ability. Better placement and more readily available help may have decreased the number of students struggling in the classes. It is possible that the more uniform distribution seen here were because such retention efforts aided a large number of the students who might have fallen in the lower end, alleviating the bimodal effect.

As a retrospective study using data from a specific sample of students, issues of generalizability are also a natural concern. Further research should explore the influence of contextual and situational factors on student performance across course sequences. Performance data ought to streamline the introductory computer science course sequence to ensure students are being properly assessed and prepared for subsequent study.

REFERENCES

Ahadi, A., & Lister, R. (2013). Geek genes, prior knowledge, stumbling points and learning edge momentum: parts of the one elephant? In *Proceedings of the ninth annual international ACM conference on International computing education research* (pp. 123-128). ACM. https://doi.org/10.1145/2493394.2493416

Alturki, R. (2016). Measuring and Improving Student Performance in an Introductory Programming Course. *Informatics in Education*, *15*(2), 183-204. https://doi.org/10.15388/infedu.2016.10

Basnet, R. B., Doleck, T., Lemay, D. J., & Bazelais, P. (2018). Exploring Computer Science Students' Continuance Intentions to Use Kattis. *Education and Information Technologies, 23*(3), 1145–1158. https://doi.org/10.1007/s10639-017-9658-2

Bornat, R. (2014). *Camels and humps: a retraction*. Retrieved on November, 2017 from http://www.eis.mdx.ac.uk/staffpages/r_bornat/papers/camel_hump_retraction.pdf

Brown, J. D. (2014). Differences in how norm-referenced and criterion-referenced tests are developed and validated? *Shiken, 18*(1), 29-33.

Burning Glass. (2016). *Beyond Point and Click: The Expanding Demand for Coding Skills* (pp. 1-12). Retrieved from http://burning-glass.com/wp-content/uploads/Beyond_Point_Click_final.pdf

Caspersen, M. E., Larsen, K. D., & Bennedsen, J. (2007). Mental models and programming aptitude. In *Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (206-210). New York, NY: ACM. https://doi.org/10.1145/1268784.1268845

Corney, M. W. (2009). Designing for engagement: Building IT systems. In *ALTC First Year Experience Curriculum Design Symposium*. Queensland University of Technology, Brisbane.

Dehnadi, S., & Bornat, R. (2006). *The camel has two humps*. Middlesex University Working Paper. Retrieved on November 2017, from http://www.eis.mdx.ac.uk/research/PhDArea/saeed/paper1.pdf

Dishman, L. (2016). *Why Coding Is Still The Most Important Job Skill Of The Future*. *Fast Company*. Retrieved from https://www.fastcompany.com/3060883/why-coding-is-the-job-skill-of-the-future-for-everyone

Dunn, L., Parry, S., & Morgan, C. (2002) Seeking quality in criterion referenced assessment. In *Learning Communities and Assessment Cultures Conference, EARLI Special Interest Group on Assessment and Evaluation*, University of Northumbria, UK. Retrieved from http://www.leeds.ac.uk/educol/documents/00002257.htm

França, A. C. C., da Cunha, P. R., & da Silva, F. Q. (2010). The Effect of Reasoning Strategies on Success in Early Learning of Programming: Lessons Learned from an External Experiment Replication. In *14th International Conference on Evaluation and Assessment in Software Engineering (EASE).* Keele University, UK.

Hartigan, J., & Hartigan, P. (1985). The Dip Test of Unimodality. *The Annals of Statistics*, *13*(1), 70-84. https://doi.org/10.1214/aos/1176346577

Höök, L. J., & Eckerdal, A. (2015). On the bimodality in an introductory programming course: An analysis of student performance factors. In *Learning and Teaching in Computing and Engineering (LaTiCE), 2015 International Conference on* (pp. 79-86). IEEE. https://doi.org/10.1109/LaTiCE.2015.25

Kafai, Y., & Burke, Q. (2013). Computer Programming Goes Back to School. *Phi Delta Kappan*, *95*(1), 61-65. https://doi.org/10.1177/003172171309500111

Lister, R. (2010). Computing Education Research: Geek genes and bimodal grades. *ACM Inroads*, *1*(3), 16. https://doi.org/10.1145/1835428.1835434

Lung, J., Aranda, J., Easterbrook, S., & Wilson, G. (2008). On the difficulty of replicating human subjects studies in software engineering. In *Proceedings of the 30th International Conference on Software Engineering* (ICSE '08). New York, NY: ACM. https://doi.org/10.1145/1368088.1368115

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

Ma, L., Ferguson, J., Roper, M., & Wood, M. (2011). Investigating and improving the models of programming concepts held by novice programmers. *Computer Science Education*, *21*(1), 57-80. https://doi.org/10.1080/08993408.2011.554722

Ott, C., Robins, A., Haden, P., & Shephard, K. (2015). Illustrating performance indicators and course characteristics to support students' self-regulated learning in CS1. *Computer Science Education*, *25*(2), 174-198. https://doi.org/10.1080/08993408.2015.1033129

Patitsas, E., Berlin, J., Craig, M., & Easterbrook, S. (2016). Evidence that computer science grades are not bimodal. In *Proceedings of the 2016 ACM Conference on International Computing Education Research* (pp. 113-121). ACM. https://doi.org/10.1145/2960310.2960312

Qian, Y., & Lehman, J. (2017). Students' Misconceptions and Other Difficulties in Introductory Programming. *ACM Transactions on Computing Education*, *18*(1), 1-24. https://doi.org/10.1145/3077618

Robins, A. (2010). Learning edge momentum: a new account of outcomes in CS1. *Computer Science Education*, *20*(1), 37-71. https://doi.org/10.1080/08993401003612167

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, *13*(2), 137-172. https://doi.org/10.1076/csed.13.2.137.14200

Sadler, R. D. (2005). Interpretations of Criteria-Based Assessment and Grading in Higher Education. *Assessment and Evaluation in Higher Education, 30*(2), 175-194. https://doi.org/10.1080/0260293042000264262

Thompson, C. (2018). *The Next Big Blue-Collar Job Is Coding*. *WIRED*. Retrieved from https://www.wired.com/2017/02/programming-is-the-new-blue-collar-job/

Watson, C., & Li, F. W. (2014). Failure rates in introductory programming revisited. In *Proceedings of the 2014 conference on Innovation & technology in computer science education* (pp. 39-44). ACM. https://doi.org/10.1145/2591708.2591749

Wray, S. (2007). SQ minus EQ can predict programming aptitude. In *Proceedings of the PPIG 19th Annual Workshop, Finland* (Vol. 1, No. 3).

Zingaro, D. (2015). Examining Interest and Grades in Computer Science 1. *ACM Transactions on Computing Education*, *15*(3), 1-18. https://doi.org/10.1145/2802752

http://www.ejmste.com