Informatics in Education, 2016, Vol. 15, No. 2, 183–204

DOI: 10.15388/infedu.2016.10

Measuring and Improving Student Performance in an Introductory Programming Course

Raad A. ALTURKI
Department of Computer Science, Al-Imam Mohammad Ibn Saud Islamic University
P.O.Box: 5701, Riyadh:11432, Saudi Arabia
e-mail: ralturki@ccis.imamu.edu.sa

Received: September 2015

Abstract. Students' performances in introductory programming courses show large variation across students. There may be many reasons for these variations, such as methods of teaching, teacher competence in the subject, students' coding backgrounds and abilities, students' self-discipline, the teaching environment, and the resources available to students, all of which can affect student performance and outcomes. Our observations in teaching programming courses (at Al-Imam Muhammad Ibn Saud Islamic University in Riyadh) are that many students (up to 50% per course) drop out. There is a strong belief by many instructors that such a high dropout rate is due, at least in part, to students underestimating the effort needed to finish this course and not following instructions as recommended. This paper reviews the factors that affect student performance in an introductory programming course (CS1) and aims to discover correlations between various assessment methods, students' participation and their final performance measured. It analyses mark distributions across various assessment methods to identify which assessment method best predicts final exam marks and overall marks, and gives recommendations for assessment in introductory programming courses.

Keywords: teaching programming, computer science education, evaluation and assessment methodologies, marks distribution, motivation, analysis.

1. Introduction[1]

It has been observed by instructors of the introductory programming course (CS1) in the College of Computer and Information Science at Al-Imam Muhammad Ibn Saud Islamic University in Riyadh (henceforth, Al-Imam University) that there is a great deal of variation in students' performance. Only a few students obtain full marks while many fail to pass or drop the course (up to 65%). After reviewing the literature, we have found that

[1] Non-standard abbreviations used in this paper: CS1 – the introductory programming course; MT – midterm; ACT: American College Testing.

the problem is not merely a local challenge at our institution, but rather a universal phenomenon (Corney, Teague, & Thomas, 2010; Hu, Winikoff, & Cranefield, n.d.; Nikula, Gotel, & Kasurinen, 2011). For example, (Nikula *et al.*, 2011) reported that more than 30% of computer science students worldwide in 1999 dropped or failed the introductory programming course. Corney *et al.* (2010) linked failure in introductory programming courses to degree withdrawal. These high dropout and failure rates should be a big concern for institutions, instructors and students. Students need to pass the course, which is a pre-requisite for all other CS courses, and the delay in graduating resulting from failure could be troublesome. Instructors spend considerable time and effort with students in lectures and labs to make programming better understood; however, such efforts may be considered ineffective considering the high rates of failure and drop out. Colleges and departments allocate resources (including instructors, rooms and lab facilities) to courses and high rates of dropout and failure use up resources that could presumably be put to better use elsewhere. To address and overcome these issues, it is of vital importance to identify factors that contribute to these outcomes.

In previous research, low success rates in introductory programming have been attributed to many factors, including student motivation (Corney *et al.*, 2010; Nikula *et al.*, 2011; Teague & Roe, 2008), the intrinsic difficulty of programming and the complexity of course structure (Nikula *et al.*, 2011), teaching and learning styles (Benford & Gess-Newsome, 2006; Corney *et al.*, 2010; Hoda & Andreae, 2014; Keen & Etzkorn, 2009; Robins, 2010; Wiedenbeck, LaBelle, & Kain, 2004). Suggestions to overcome such problems and improve success rates include using less complex course structures (Keen & Etzkorn, 2009; Nikula *et al.*, 2011), employing more collaborative learning (Hanks, 2005; Mcdowell, Werner, Bullock, & Fernald, 2003; Teague & Roe, 2008; Wood, Parsons, Gasson, & Haden, 2013), redesigning course material (Corney *et al.*, 2010) and improving teaching style (Hoda & Andreae, 2014; Robins, 2010). Researchers have established that failure is not always associated with cognitive ability, but sometimes with motivation and teaching style (Teague & Roe, 2008).

We believe that motivation is a key to successfully passing introductory programming, along with proper guidance; it is the key to improving other shortcomings. Motivation is very hard to measure objectively, but success or failure to pass a course may be seen as a reflection of motivation. The most reliable way to get relatively objective measurement is to use students' marks before and during the course. In CS1 courses, students are assessed and graded using various instruments: midterm (MT) exams, quizzes, assignments, projects, attendance, lab exams and final exams. Using these assessment items to evaluate overall achievement is very important for predicting student performance and providing relevant support. Previous marks gained at university, high school marks and standardised tests can help predict performance as well. In this study, we wish to see if the mark distribution of assessment methods currently used in the basic programming course are the best ones to reflect student performance and improve their motivation.

In fact, many indicators can be used to predict performance in introductory programming courses. The literature in this area has covered various such indicators such as: standardised tests (e.g. the ACT) (Brown, 2013; Butcher & Muth, 1985), high school

grades (Bergin & Reilly, 2005; Werth, 1986), Grade Point Average (GPA) (Terry R. Hostetler, 1983), previous courses taken (Brown, 2013; Campbell, 1984; Werth, 1986), prior experience using computers (Wilson, 2002), English and mathematics background (Butcher & Muth, 1985; Konvalina, Wileman, & Stephens, 1983; Pillay & Jugoo, 2005; Wilson, 2002), overall student comfort and confidence levels (McGill, Volet, & Hobbs, 1997; Wilson, 2002), and customised tests (Bergin & Reilly, 2005; Farrell, 2006; Kurtz, 1980; Porter & Zingaro, 2014). Other studies have shown that student behaviour during the course is a strong indicator to their performance. For example, The two studies (McGill *et al.*, 1997; Watson, Li, & Godwin, 2014) have shown that time spent in the computer lab has a strong correlation with the final grade. (Denny, Luxton-Reilly, Hamer, Dahlstrom, & Purchase, 2010), showed that students who attend more quizzes have higher exam scores than those who attend fewer.

In the introductory programming course, there is also variation among different assessment methods for the same student. For example, a student could have a high mark on an assignment and a low mark on the midterm, which can be a sign of something wrong. We wanted to know if some assessment methods used in the CS1 course are better or worse for measuring a given student's ability and creating motivation to excel. Moreover, different assessment methods have different weights from the overall total mark; thus, specifying the right weight for each assessment method is vital for measuring a student's ability and boosting their motivation.

The introductory programming course at Al-Imam University is taught in the first semester of the degree programme, after passing the preparatory year, and is required for all majors related to computing, including computer science, information systems, and information management. CS1 has also been introduced to the curriculum of other degrees in engineering and mathematics. According to the course description, the main goal of this course is to introduce problem-solving and programming principles to students using C++. The methods of assessment used in the course include written exams (e.g., quizzes and midterms) and practical exercises (e.g., lab exams and assignments).

Instructors normally associate failure and dropout with weak background or lack of self-discipline on the student's part and assessment methodology is always a debate among instructors as there is no clear evidence as to the best assessment method for measuring students' abilities in programming. Providing extra counselling and support for students at risk is very important; therefore, having a way to predict student performance is essential for such a process. In Section 2, we review factors that affect student performance in introductory programming followed by a detailed analysis of students' marks in the CS1 course across four consecutive terms with different instructors in Section 3.

2. Factors Affecting Performance

This section reviews the literature on factors that affect the performance of students in introductory programming. Literature and research in this area can be classified into two main categories: research that focuses on 1) factors associated with students and 2) factors associated with instructors and/or the learning environment. The factors associated

with students are all factors that students have or control like learning style, background and cognitive ability. Factors associated with instructors and the environment are those factors that instructors have or control, or relate to the teaching environment includes teaching style, complexity of course set-up, assessment methods used, teaching room facility, other students' competence level and course description.

2.1. *Factors Associated with Students*

Previous studies on factors associated with student performance in introductory programming include research on student motivation (Corney *et al.*, 2010; Nikula *et al.*, 2011; Teague & Roe, 2008), learning style (Benford & Gess-Newsome, 2006; R. M. Felder & Spurlin, 2005; Hoda & Andreae, 2014; Keen & Etzkorn, 2009; McGill *et al.*, 1997; Robins, 2010; Thomas, Ratcliffe, Woodbury, & Jarman, 2002; Wiedenbeck *et al.*, 2004), prior background and experience (Butcher & Muth, 1985; Konvalina *et al.*, 1983; Pillay & Jugoo, 2005; Wilson, 2002) and student comfort and confidence level (McGill *et al.*, 1997; Wilson, 2002). A brief description of major work in each of these areas is given in this section.

2.1.1. *Motivation*

Previous studies in the correlation between student performance and their motivation have shown a strong connection. For example, Corney *et al.* (2010) redesigned the CS1 introductory programming course to be more engaging and collaborative in order to motivate students. They used a pair-programming (programming in pairs of students; PP) method during lab sessions, and redesigned the course to provide more generative information and engaging content about various technologies like databases, the Web, and networking. The main goal was to make programming more interesting and more relevant to real-world problems. The changes resulted in higher success rates and less dropout; further, they were geared to what industry requires from graduates (to be good communicators and team players who are business-minded. The feedback on these amendments from their students showed the importance of making programming relevant to real problems.

High failure and dropout rates were also observed in introductory programming classes at Lappeenranta University of Technology in Finland (Nikula *et al.*, 2011). There were three possible causes investigated of the high failure rate: the difficulty of programming, course complexity, and student motivation. Motivation was investigated using two-factor theory, where factors causing satisfaction (motivators) and dissatisfaction (hygiene factors) were applied to students in the course. The results show that students with good intrinsic motivation had less difficulty completing the course than other students; the problem arose with students who had poor motivation, and it was found that these students were enrolled in the course rather arbitrarily and faced few consequences for failure. The study concludes that low motivation and immature behaviour is a major problem that affects pass rate. Three steps were introduced to tackle the problem and increase motivation as following: 1) reduce de-motivators (e.g., the complexity of the

course setting), 2) increase intrinsic motivators by making course assignments and projects more appealing and interesting to students, and 3) introduce extrinsic motivators like asking students to submit sections of assignments weekly rather than all at once at the end, along with project progress reports . After these improvements, the pass rate increased from 44% in 2005 to 68% in 2009.

Employing a more collaborative and engaging environment is key in improving success rate in CS1 course. A survey of difficulties faced during a programming course at the University of Queensland in Brisbane (Teague & Roe, 2008), focusing on motivation and social issues was conducted. The survey showed that at week 8 of the term, 48% of students thought the course was more difficult than expected and most students agreed that employing collaborative learning would make the course more engaging and interesting which would help their motivation.

2.1.2. *Learning Style*

Learning style is the method by which students prefer to acquire and process knowledge. Some students prefer very detailed explanation by the teacher, while others prefer more general or abstract explanation first, followed by details as necessary. Some students prefer to get information as graphic representations where others prefer it to be verbal. The problem happens when there is a mismatch between studying and teaching style as the instructor teaches in a style that is different than the recipient is expecting or prefers the most. In fact, there are many theories and models about learning styles. One of many models of learning style, proposed by (R. Felder & Silverman, 1988) where learning styles can be modelled in four main dimensions: 1) active/reflective, 2) sensing/intuitive, 3) visual/verbal, and 4) sequential/global. Any given person's style is a mixture of the four dimensions, with a different density for each dimension; a person can be of more as an active learner than all other traits and can be balanced between any two or more traits. (R. M. Felder & Spurlin, 2005) stressed that this model cannot predict performance but is intended instead to 1) guide instructors to satisfy the needs of diverse students with different learning styles and 2) help students understand their own their learning styles and use them more effectively to learn better.

Thomas *et al.* (Thomas *et al.*, 2002) have shown that learning styles correlate with student performance. They studied the impact of preferred learning styles on the performance of students at the Aberystwyth University in Wales. They used the same model as (R. Felder & Silverman, 1988) and found that the current system of teaching at their university is more suitable for reflective, verbal and sequential learners while active, sensing, and visual learners are disadvantaged by it. The study concluded that if the current method of teaching is used, the performance of students could be predicted unless they employ other methods that satisfy the need for all type of learners. They developed a new environment to care for all type of learners, called MAP (Monitoring, Assessment and Provision) where different students can make the most of the resources depending on their types of learning. They also noted that the combination of student autonomy and teacher support in CS1 course is backed by other work (Tanner & Jones, 2000)

Mental models and self-efficacy are considered important parts of student learning style with clear impacts on student performance in any programming course. A mental

model is the representation of concepts and ideas related to a given area, such as programming, in one's mind; and self-efficacy is a person's confidence in his or her own ability to succeed in a given context. Wiedenbeck *et al.* (Wiedenbeck *et al.*, 2004) found that a mental model has an impact on students' success and can improve their self-efficacy. Their study suggests to support the good mental model of students to make programming easier to be thought of. One way of supporting the development of a mental model that they mention is to use exercises that trace the logic of the program, which has been shown to be effective in developing a mental model. Also, the study suggests encouraging self-efficacy by providing course content that matches students' performance level rather than their level of confidence.

Another way to look into learning style is illustrated by Robins in (Robins, 2010) where an illustration of high failure in introductory programming course is shown and explained in terms of the learning edge momentum (LEM) effect. Learning Edge Momentum's (LEM) main idea is to associate student ability to grasp any new concept in programming to the last related concept the student learned. That is, students cannot grasp advanced topics if prerequisite topics are not understood, especially in programming, which is a very interconnected field. This also helps explain the finding of (Thomas *et al.*, 2002) showing students with sequential learning styles to be much better learners than other students. Teaching students about this effect will help them understand how to manage their studies, which require continuous revision of concepts covered to master them before taking on new concepts.

2.1.3. *Prior Background and Experience*

Student background plays a vital part in understanding programming concepts and succeeding to pass the course. As noted above, researchers have found that previous experience using computers (Wilson, 2002), programming, problem-solving, and mathematical and English experience have big impacts on success in programming (Campbell, 1984; Konvalina *et al.*, 1983; Pillay & Jugoo, 2005; Wilson, 2002). For example, an early study (Campbell, 1984) showed that providing a preliminary course in programming led to a significant difference in performance between students who finished that course and students who did not. It is not clear if the preliminary course was a filter rather than a way to improve and prepare students, as the preliminary course had a very high failure rate, and as a result, students with poor performance in this preliminary course may have been able to change their majors and not take the CS1 course. Other research (Konvalina *et al.*, 1983) supports this notion of the background effect on programming performance after studying the measurable factors that affect the withdrawal rate. Their results show correlations between certain factors (age, high school marks, mathematical experience, and programming experience) and the withdrawing status of students. Their results were used to build a predictor model that guided students' placement in the CS1 course, resulting in a significant decrease in the withdrawal rate, from 40% to 23%.

Other work supports the correlation between students backgrounds and performance in programming is reported by Wilson (Wilson, 2002) . The study was conducted to identify factors that affect student performance in introductory programming

CS1 and in particular, to determine which factors could be utilized to increase female involvement in computer science. The study revealed that mathematical background has a significant correlation with results in the programming course, but that previous computer experience (including programming and videogame playing) did not significantly affect the overall results in the programming course. Similarly, (Pillay & Jugoo, 2005) tried to determine the effect of students' problem-solving ability and language skills on their performances in the CS1 course, and showed that having taken previous courses in mathematics and problem-solving significantly and positively affected performance, and that students whose mother tongue was the language of instruction performed much better.

2.1.4. *Comfort and Confidence Level*

The comfort and confidence of a student studying programming have an interesting effect on overall performance. Wilson (Wilson, 2002) conducted a study to see what factors could affect CS1 students' performance in order to increase female involvement in computer science. The study revealed that student comfort level to be the most significantly correlated with student performance (interestingly, much higher than math background). Comfort level was measured in the study using several indices as following: "(1) likelihood of asking questions/answering questions in class, (2) likelihood of asking questions in lab, (3) likelihood of asking questions during office hours, (4) perceived anxiety while working with the computer on programming assignments, (5) perceived difficulty of the class, (6) perceived difficulty of writing computer programs in general, and (7) perceived understanding of concepts in class compared to classmates". (Wilson, 2002) Although comfort level had a high correlation with performance, a causal relation cannot be inferred, as students' comfort level could be a result of their ability and their understanding of the course. Another study (McGill *et al.*, 1997) was conducted to investigate success factors in a distance learning programming course. The study asked students questions at the beginning of the course to identify different aspects of their backgrounds and perceptions, revealing that confidence in one's ability to complete the course has a significant correlation with completion rate especially in distance learning, where effective teacher support may be more difficult.

2.2. *Factors Associated with Instructors or the Learning Environment*

Factors associated with instructors include teaching style (Benford & Gess-Newsome, 2006; Hanks, 2005; Hoda & Andreae, 2014; Mcdowell *et al.*, 2003; Robins, 2010; Wood *et al.*, 2013), course and material settings (Keen & Etzkorn, 2009; Pears *et al.*, 2007) and instructor ethnicity and interaction (Benford & Gess-Newsome, 2006). A brief description of major works in these areas is reviewed in this section.

2.2.1. *Teaching Style*

Teaching style is the way that an instructor delivers or transfers knowledge to students. Teaching styles vary across individual instructors, institutions, and countries. Variation

in teaching styles and course delivery methods has a clear impact on student performances. In a recent work (Vihavainen, Airaksinen, & Watson, 2014), it was found that interventions in teaching styles improved success rates by one-third on average, and that interventions that employ collaboration and peer support were the most effective in improving pass rates. Another study (Benford & Gess-Newsome, 2006) shows a clear connection between teaching style and student performance in gateway courses. Teaching style for each course was given a Reformed Teaching Observation Protocol (RTOP) score between 0 and 100 to see how traditional/didactic or progressive the teaching style used in the delivered courses. Low score means a traditional and didactic teaching method is used while high score means a more activity based and progressive method is used. The research found that there is a clear connection between the average RTOP score and the average student's achievement.

Robins (Robins, 2010) presents a different view, explaining high failure rates with reference to the learning edge momentum (LEM) effect, arguing that instructors who understand this effect and help students with poor results will see better results. This effect could also explain other reported work about the good performance of students with sequential learning style in CS1 course(Thomas *et al.*, 2002). Other reported work (Campbell, 1984) show that taking a CS0 course before the CS1 can improve performance in the latter, while Hoda and Andreae (Hoda & Andreae, 2014) present findings showing that a teaching strategy they developed for the introductory course at Victoria University in Wellington reduced failure rates from 45% to 35%. The strategy developed based on minimizing complexities and dependences in early topics covered, maximizing the chances of mastering these concepts and skills and providing opportunities for early recovery.

Collaboration in teaching programming has received good attention from researchers, who have found that it has quite a strong positive effect on improving students' performances. One well-reported collaboration technique in teaching programming is pair programming (PP). PP allows two people to solve or write a program collaboratively by sitting beside each other at the same workstation; one person works as a 'driver', handling keyboard and mouse in order to write the code, and the other as an observer who check the code and give live feedback to the 'driver'. Pair programming has been shown to be an excellent methodology for teaching introductory programming course (Hanks, 2005; Mcdowell *et al.*, 2003; Wood *et al.*, 2013) as it makes students more engaged and more motivated (Wood *et al.*, 2013). Previous work in PP has shown an increase in student confidence (Hanks, 2005; Wood *et al.*, 2013), save time (Wood *et al.*, 2013) and contribute in making better programs (Mcdowell *et al.*, 2003). As a result, PP is contributing to better pass rates (Hanks, 2005; Mcdowell *et al.*, 2003) and attracting more students to majoring in computer science (Mcdowell *et al.*, 2003). One limitation of PP is the need for two students to be located in the same place; however, one study has shown that results similar to those of PP implementation can be achieved using 'distributed pair programming' where students work in pairs remotely (Hanks, 2005). A recent analysis (Vihavainen *et al.*, 2014) has shown that pair programming can improve pass rates by at least 10%. Having said that, when comparing pair programming technique solely with other techniques, pair programming could not solve all

programming difficulties as any interventions to the programming course should have several dimensions that improve current problems.

Peer Instruction (PI) is a pedagogy technique that is used to make students more engaged in the course by triggering their self-learning instinct by firstly asking question followed by peer discussion and then correcting the answer by instructor. PI was firstly used in physics (Crouch & Mazur, 2001){Formatting Citation} and then it has become common in other subjects like biology and recently in computer science teaching (Porter, Bailey-lee, & Simon, 2013). There have been many studies to show how incorporating such technique can affect students' performance positively in teaching programming. For example, a study by (Beth Simon, Parris, & Spacco, 2013) showed an increase in final exam score by around 6% when PI is used. Other study by (Chase & Okie, 2000) has shown a dramatic decrease in the dropping and failure rate when PI method were used. Leo Porter *et al.* (Porter *et al.*, 2013) explains that incorporating PI method in teaching CS1 course has resulted in reducing failure rates on average by 67% (23% reduced to 8%). PI is not only improving results but also increase satisfaction among student and instructor when teaching programming. (B. Simon, Kohanfars, Lee, Tamayo, & Cutts, 2010).

2.2.2. *Course Set-Up and Environment*

Lecture notes and course set-up complexity have considerable impact on student performance in the programming course. Keen and Etzkorn (Keen & Etzkorn, 2009) have conducted a case study to see how the complexity of lecture notes can affect student performance in programming course; complexity is measured by the number of technical terms from the field of computer science divided by the total number of words in the notes has a clear correlation with students' performance. With reference to LEM effects, Hoda and Andreae (2014) have developed an effective teaching strategy that reduces the complexity of a course especially at the early stage of the term to help students master the fundamentals, as mentioned above.

Another important issue related to course set-up is programming language choice, which plays a considerable role in the way programming concepts are introduced to students. There have been many debates about the most suitable language for introductory programming instruction (usually Java, C and C++; (Pears *et al.*, 2007), but sometimes also other languages like Python and Ruby that have good affordances for novice students. The bottom line in choosing a programming language should always be how it relates to the course objectives and curriculum goals. The real problem is how to make teaching the chosen language as simple as possible, as many instructors tend to teach the minor details and forget the major objectives of the course.

Class and university environment have also been reported to affect student performance. A study by Benford and Gess-Newsome investigated the factors that could affect student performance in gateway courses. Interestingly, the study has shown a clear gap among different ethnicity groups and their averaged GPA. When comparing averaged GPAs, International students are the highest and African American and Native American are the lowest. It has been discussed that such variation could be for many reasons, include student preparedness and faculty diversity.

3. Methods

The goal of this study is to discover any correlation between marks on different assignments and tests within the programming course. The marks of 138 students were analysed across different sections, four terms, and instructors. The course description of the introductory programming course (CS140) at Al-Imam University gives a breakdown of marks across written tests (e.g. Quizzes, Midterm and Final) and coding tests (e.g. Lab exams and assignments). Written tests are paper based and question types include asking student write a code, correct a code and trace a code. Coding exercises is computer based and normally ask student to write code or function and shows how it compiles and run. The weight of each mark – of these assessment methods – can be varied from one time to another. The weights of each assessment method used in the course for two periods in the last four terms are given in Table 1 where changes were done in the Lab and assignments weights. As can be seen in Table 1, 60%–70% of total marks are given to written tests and 30%–40% to practical programming exercises; the latter help students gain a better understanding of programming. That explains why separate assessment methods to evaluate student participation and progress in practical programming are always part of programming course. In our study, we wanted to test this notion as variation of students performance across assessment methods is observed. Thus, this analysis aims to investigate how marks in exercises, exams, and participation correlate with each other, which assessment methods are most effective in measuring student performance, and what is the best weight for each method.

In order to compare different marks for the assessment methods that have different weights, marks were standardized as percentages for analysis. For example, if quizzes were marked to have 10 as the highest mark, quizzes marks gained will be divided by 10 and then multiplied by 100 in order to have a percentage number between 1–100%. An example of that is illustrated in the table below (Table 2) which shows the marks for 3 students as numbers (to the left) and as percentages (to the right) for four assessment methods.

After marks were standardized as percentages for analysis, we compared marks by subtracting the percentage of each mark with all other marks for the same student which resulted in having a percentage number that shows how a specific mark varies across

Table 1
The weight of marks in the programming course

Assessment method	Period 1	Period 2
Quizzes	10%	10%
Assignments	10%	0%
Midterm (MT)	20%	20%
Lab	20%	30%
Final Exam	40%	40%
Total	**100%**	**100%**

Table 2
Illustration of method used to weight marks

Student	Marks as number				Marks as Percentage			
	Quiz (max 10)	Assign-ment (max 10)	Midterm (MT) (max 20)	Final (max 60)	Quiz	Assignment	Midterm (MT)	Final
A	**9.5**	**10**	**10**	**15**	**9.5** / 10 x 100 = 95%	**10** / 10 x 100 = 100%	**10** / 20 x 100 = 50%	**15** / 60 x 100 = 25%
B	6	7	8	20	60%	70%	40%	33%
C	5	5.5	14.25	41	50%	55%	71%	68%

other marks. Also we have averaged the variation of each mark with other marks for all students, and obtained the average deviation to see how marks deviate from the average of the variation. Also, we determined whether variations were positive (indicating over-assessment) or negative (under-assessment).

An example of how variations were calculated for quizzes in comparison to other marks is shown in Table 3 below. The data in Table 3 below is made-up for three students and does not represent real data. The table first represents the marks obtained on four assessment methods for three students (A, B, and C), as percentages. The marks in percentages in the first four data columns are taken from Table 2. Then, the last three columns show variation of other marks in comparison to quizzes. For example, for student A, the quizzes mark is lower than the assignments mark by 5% and higher than the midterm and the final by 45% and 70% respectively. Student C data show different results: the quizzes mark is lower than the assignments, midterm, and final marks by 5%, 21%, and 18% respectively. The last two rows of the table calculate the average and average deviation of the absolute values of the first three rows. The higher the average of the deviation, the greater is the gap between marks, and the lower the deviation of the variation, the greater is the correlation with the quizzes. The data illustrated in the table shows correlation between assignments and quizzes marks.

Table 3
Example of marks variations from quizzes

Student	Mark as Percentage				Variation from Quizzes		
	Quiz	Assign-ment	Midterm (MT)	Final	Assignments	Midterm (MT)	Final
A	95%	100%	50%	25%	95-100=**-5%**	95-50=**45%**	95-25=**70%**
B	60%	70%	40%	33%	60-70=**-10%**	60-40=**20%**	60-33=**27%**
C	50%	55%	71%	68%	50-55=**-5%**	50-71=**-21%**	50-68=**-18%**
Average of absolute value	51%	56%	40%	32%	5%	22%	29%
Average deviation of absolute value	26%	29%	20%	19%	3%	12%	21%

4. Results and Discussion

The analyses of student marks in the CS1 programming course offered at Al-Imam University is showed and discussed in this section. Firstly, we analyzed the performance of students for one instructor across three terms as in section 4.1, followed by analysis of students' performances for different instructors in the same term where most of the teaching materials and assessment methods were unified as in section 4.2. We summaries our findings at the end of this section 4.3.

4.1. *One Instructor Across Terms*

This section presents and analyses student marks across three consecutive terms with the same instructor using the same teaching method and mark distribution (weighting). It compares each assessment method (assignments, quizzes, midterm exam, lab exams, and final exam) against the others, and these methods are compared with the total mark to see how they correlate. Figures Fig. 1 to Fig. 6 plot the variation of marks against each other, where the X-axis represents the fixed assessment method mark and the Y-axis shows the percentage of variation for all marks with fixed marks on the X-axis.

The variation of marks from quizzes is plotted in Fig. 1, which shows variation of four assessment methods and total marks against the quizzes marks. The general trend shows that as the quizzes marks increase, variation is also increasing. Variations start with negative values between –20 and –10% when the quizzes marks are around 0, and become positive for higher marks, between 40% and 100% variation around full marks. The negative variation mean that quizzes marks are lower than other marks and vice versa. The overall trend of the figure shows that quizzes marks are normally lower than lab marks for students who achieved less than 70% in quizzes but higher for students who achieved more than 70%. Also, the figure shows that assignments variation is highest overall, showing a range of –60 to 100%. The reason for such variation is caused by

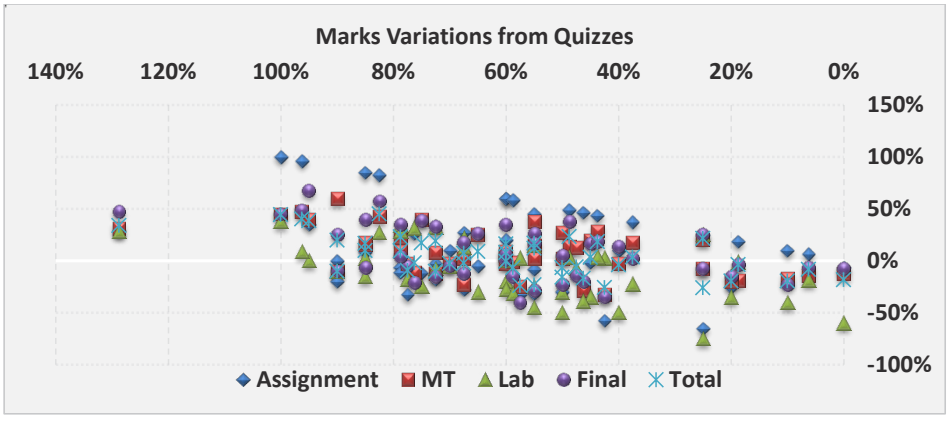

Fig. 1. Marks Variations from Quizzes.

students earning 0 in their assignment either because of not submitting or plagiarising, which entitled them to have the lowest mark of 0.

Fig. 2 plots marks variation from assignments marks. The general trend increases from negative for low assignments marks to become positive for higher marks. That means students who achieved zero or low marks in assignments are normally achieving better marks in other assessment methods. Many students achieved highly in assessment methods such as midterm and final but got zero on the assignments. That variation could give an indication of low motivation to do assignments or of the plagiarism behaviours that led to the low results. As assignment marks increase above 40%, they become higher than those for other assessment methods, which we believe is normal, as students have more time to work on the assignment and learn from fellow students how to solve it. However, this flexibility in assignments should have a higher impact, as many students probably aim to solve the assignment for the sake of marks but not for the sake of understanding and mastering programming.

The mark variation from the midterm is shown in Fig. 3, with a general increase from negative for low midterm marks to positive for higher marks. The figure shows high variation of assignment marks, which can again be attributed to the considerable number of students who achieved zero. It is interesting to note that the variation of lab marks is always negative when midterm marks are less than 55% and become positive afterwards. That means lab marks are higher than midterm marks for students who achieved less than 55% on their midterm. This could be because the lab environment is more casual than the midterm and lets students think more and more effectively, learn and get help from other students in particular, students of a low level in programming. The variation of the final exam from the midterm is lowest across all marks, which could be because of the similar nature of the two exams (written exams that explicitly test students' knowledge of certain programming concepts).

Fig. 4 plots variations from lab marks, showing a clear gap between students' marks gained in the lab. This may be due to the casual marking procedures in the lab, where marks are always rounded. It is very clear that lab marks do not correlate with other marks, as the variation is always high and inconsistent. For example, some students

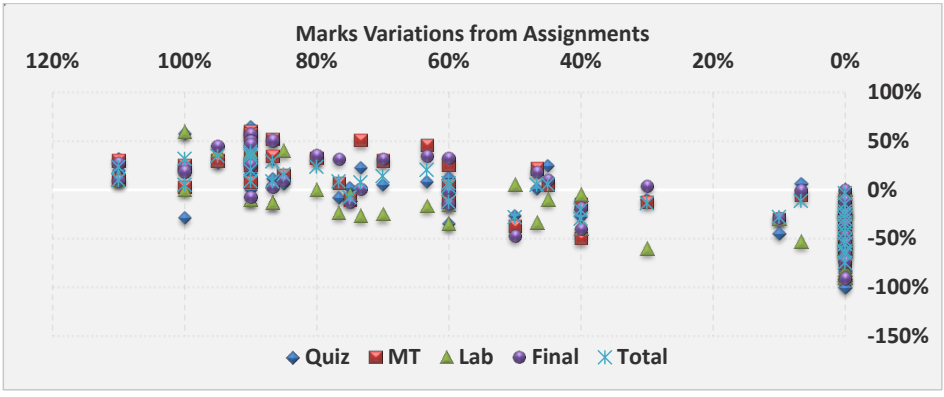

Fig. 2. Mark Variation from Assignments.

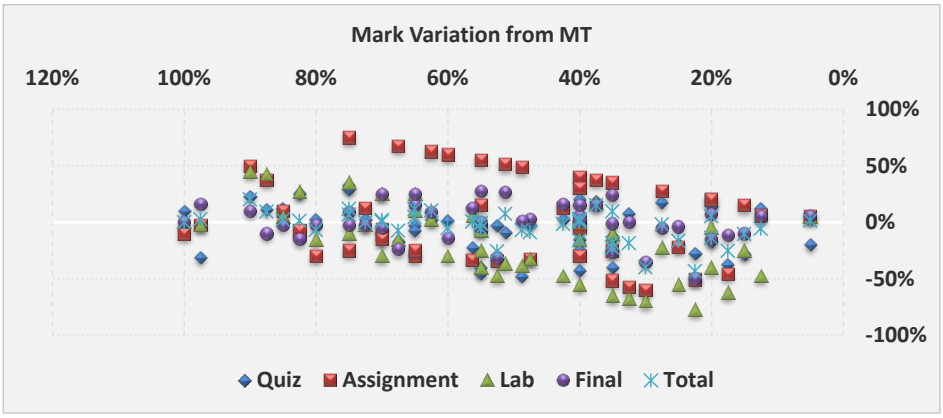

Fig. 3. Mark Variation from Midterm.

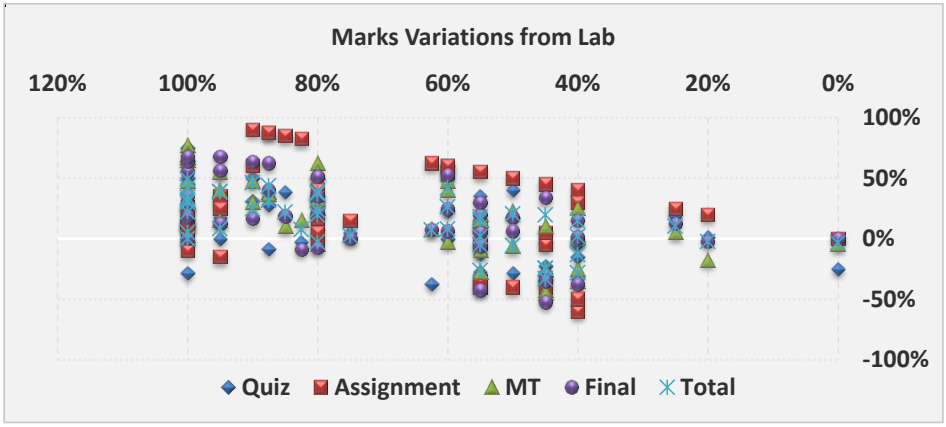

Fig. 4. Marks Variations from Lab.

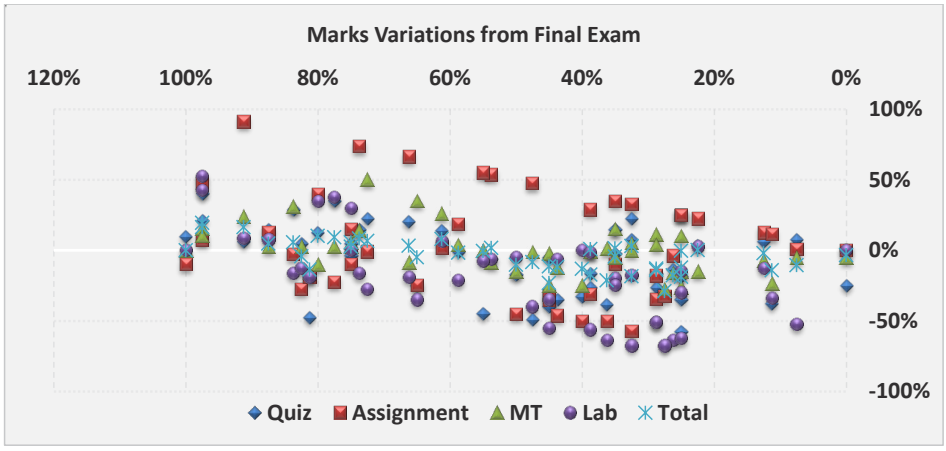

Fig. 5. Marks Variations from Final Exam.

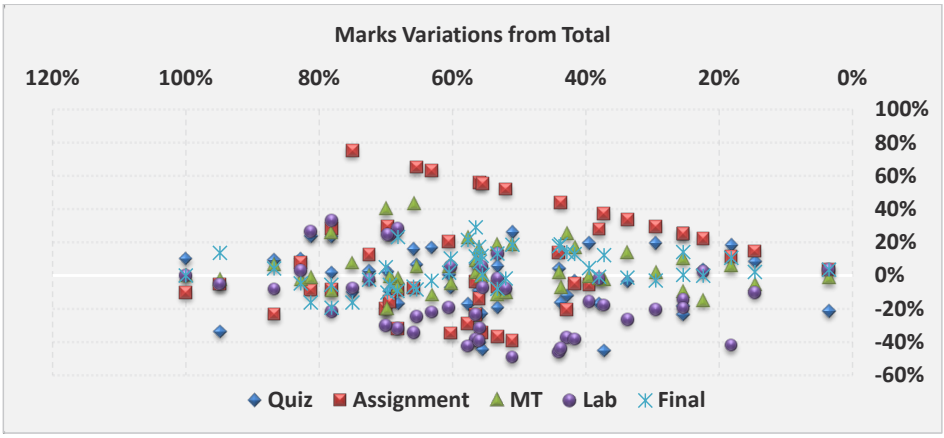

Fig. 6. Marks Variations from the Total.

achieved well in the final exam, and so the variation of their final marks from their lab marks is extremely different. This variation of marks could give an indication of the importance of the lab as a method of teaching and of making students more comfortable in studying. However, the weight of the lab should be reduced in order not to affect the real performance of students.

The variations of final exam marks from other assessment methods are plotted in Fig. 5, where a clear correlation with the total marks is clearly identified. This correlation is because the final exam accounts for a big percentage (40%) of total marks and because variations of other assessment methods cancel each other out in the total marks. The midterm is the most closely correlated assessment method with the final exam, with several cases showing final exam marks higher than midterm marks. This is logical, as normally students with low marks in midterm try to work harder after the first midterm. The highest variation from the final exam marks is in lab marks, where final marks are always lower than lab marks except for students who achieved above 80% on their final.

Fig. 6 plots the total marks earned in the course against individual assessment methods. As seen in the previous figures, the final exam and midterm are the best proxies for total marks, and labs and assignments the worst.

4.2. *Different Instructors in One Term*

This section presents and analyses students' marks across different sections of the course taught by different instructors in the same term. Major course instructions were unified by unifying exercises, quizzes, lab exams, midterm exam and final exam. As in the previous section, we compare each assessment method's marks against the others. (Note that assignments were not used this term). Then, each method's marks are compared with the total marks. Figures Fig. 7 to Fig. 11 plot the variation of marks, where the X-axis represents the fixed assessment method mark and the Y-axis shows the percentage of variation from that mark for the other assessment methods marks.

The variations of marks from quizzes are plotted in Fig. 7, where the general trend shows that as the quizzes marks increase, the variation is also increasing. Variations start with negative values between –5% and –45% when quizzes marks are around 0 and become positive for higher quizzes values, up to 88% for some students.

The overall picture of the figure show that quizzes marks are normally lower than lab marks, especially for students who achieved less than 50% in quizzes, and that this changes to –+20% variation for students with high marks in quizzes. It is very interesting to note that as quizzes marks increase, variation with the final exam increases as well. That means that students who achieved highly in their quizzes are not always doing so well in their finals. As a result, quizzes cannot be used to represent final exam results.

Fig. 8 shows variation from the midterm exam, where the midterm is always higher than the final exam and where, as the midterm mark increases, the variation increases slightly as well. The variation with lab marks is always negative, which means lab marks

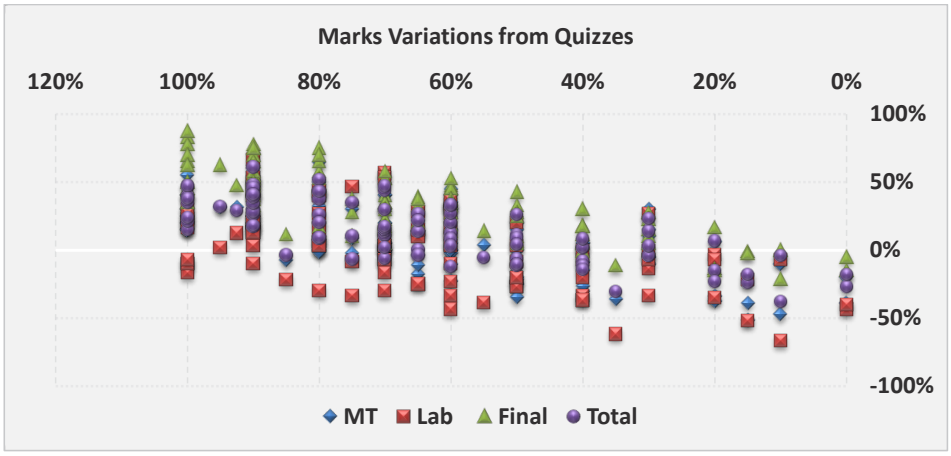

Fig. 7. Marks Variations from Quizzes.

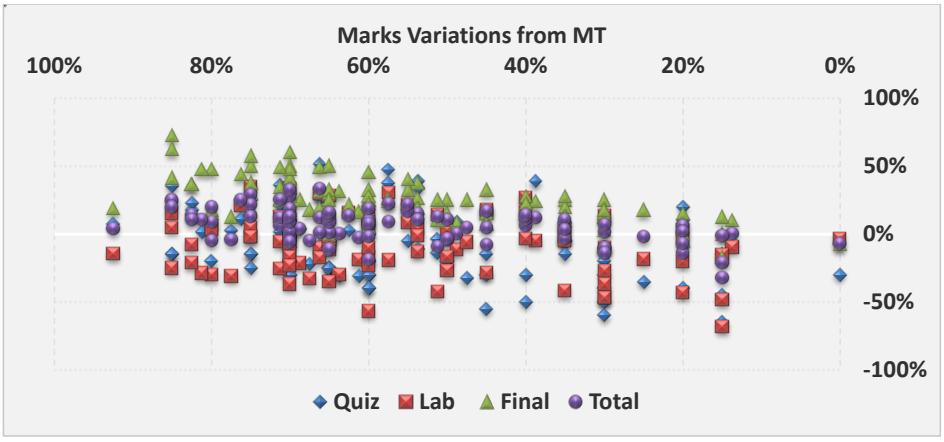

Fig. 8. Marks variations from Midterm.

in most cases are higher than midterm marks. The variation of total marks from midterm marks is the lowest across all marks; thus, overall total results can be estimated easily from the midterm results.

Fig. 9 plots the marks variations from the lab marks, where variations increase as lab marks do, from negative for low lab marks to positive for higher lab marks. Unlike other marks, there is a gap observed for the low lab marks, as lab instructors did not give low marks in most cases. This gap does not align with the other marks, for instance midterm and final marks, which are distributed across the axis. Lab marks in most cases are higher than final exam marks; this is true also for MT and quizzes marks in relation to lab marks that are higher than 83%.

Figures Fig. 10 and Fig. 11 plot variations against final exam and total marks respectively. It is clear that final exam marks correlate with total marks, which is expected, as the final exam accounts for 40% of the total. Final marks are clearly always lower than

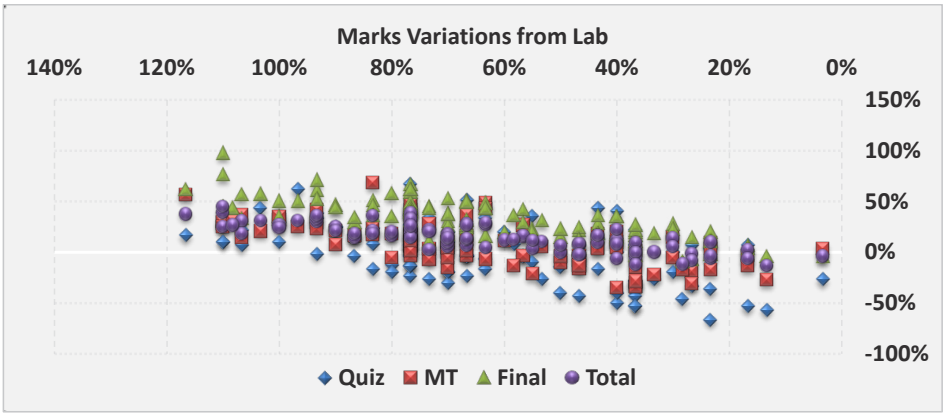

Fig. 9. Marks Variations from Lab.

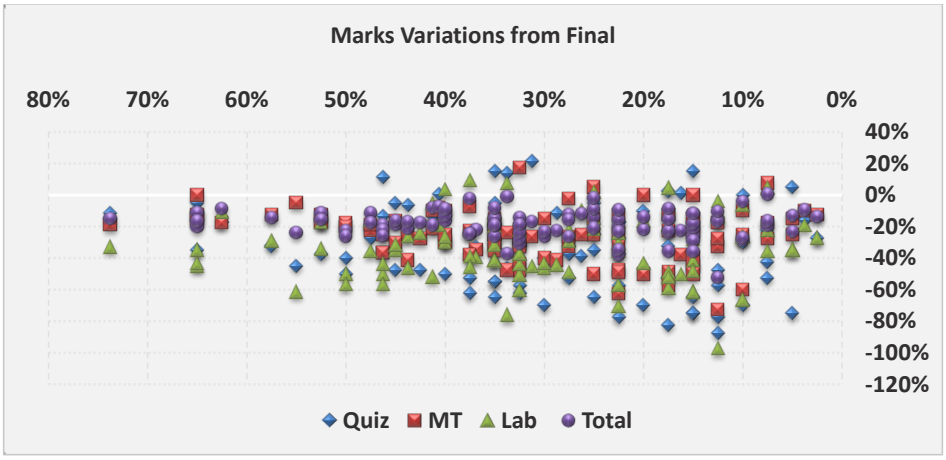

Fig. 10. Marks Variations from Final.

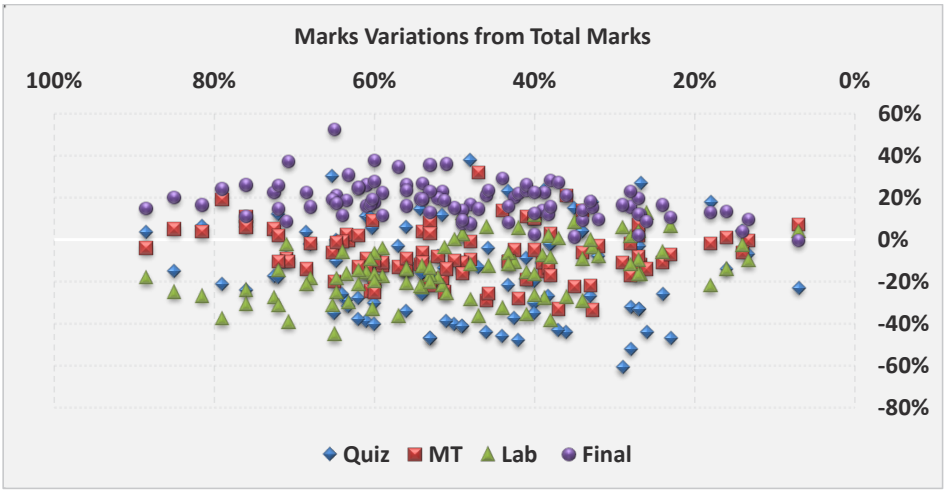

Fig. 11. Marks Variations from Total Marks.

other marks, which gives an indication of the difficulty of the exam in comparison to the other assessment methods. The midterm correlates second most closely with total marks, and labs and quizzes the least.

4.3. *Summary*

The summary of the two experiments (one Instructor across terms and different instructors in one term) is given in this section. The results show slight differences between the trends in the two data sets, which makes it hard to generalize across them in most cases. However, there is some clear evidence of important issues and of correlations between some assessment methods. Figures Fig. 12 and Fig. 13 give a summary of the results of marks analysis for both experiments combined. Fig. 12 averages the marks variation of each assessment method against all other assessment methods, while Fig. 13 averages the deviation of the averaged marks variation for each assessment method. The two figures together show that quizzes and lab marks have the highest variation and deviation from final exam marks and that in most cases quizzes and lab marks are higher than final exam marks. This is as expected, as quizzes and lab exams are normally easier than the final, relate to more recently explained material, and have a less formal structure. The midterm results in the two figures are very interesting: they show that the midterm is the best indicator for total results, as it has the least variation and deviation when compared with total marks. That is probably because the midterm is always in the middle (between finals on one hand and labs and quizzes on the other) in term of difficulty, number of topics covered, and time allotted.

The results of the first experiment, where assignments were given to students, indicate low motivation to do the assignments, as many students achieved zero scores in them but performed well on exams and in the course as a whole. The lab marks in many cases ei-

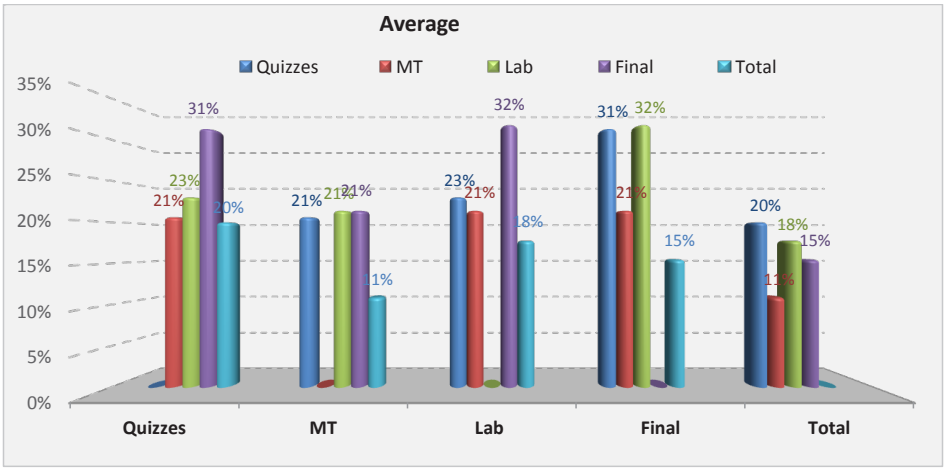

Fig. 12. Average of Marks Variations against Each Other.

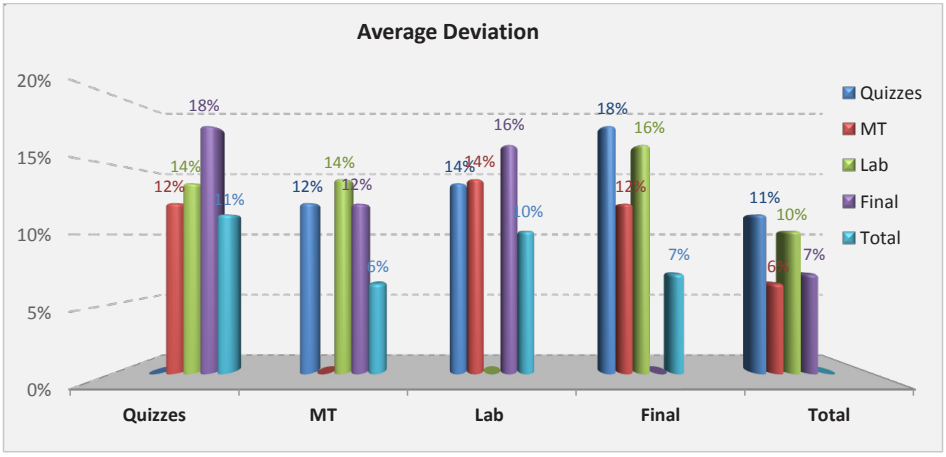

Fig. 13. Average Deviation of Marks Variations against Each Other.

ther underestimated good students or overestimated poor students, as clearly seen in Fig. 9 where many students achieved low results in lab while achieving high or good results in quizzes and other assessment methods; this indicates the importance of lowering lab weights. Lab Marks are still valuable for learning and should be retained to encourage students to attend and participate but not to not to assess their programming skills.

5. Conclusion

Previous work on factors that affect student performance in introductory programming courses clearly shows the importance of many factors that have direct or indirect effects,

including student motivation, confidence, comfort level, background and teaching style. Our analyses of marks gained during the CS1 course for different instructors across different terms show a clear correlation between certain marks within the course, mainly midterm, final and total marks. Lab and assignment marks do not correlate with other marks and are highly variable. Quizzes can be considered a slightly better way to measure student performance as they correlate slightly with total marks. Such results could be related to the studying culture in Saudi Arabia and to students' previous perceptions of studying, i.e. students normally take exams seriously, but devote less attention to assignments and lab work.

Simplifying the course structure and reducing its complexity have proven to have a positive effect on performance. Reducing complexity can be achieved by focusing on the fundamental concepts in programming and reducing the volume of course materials. In addition, students probably need to be offered simpler mark distributions (that is, fewer assessment methods) as a complicated assessment framework might distract them from the main goal of the course. We suggest reducing the weights of labs, assignments and quizzes, and increasing the weights of final and midterm exams. We plan to extend this work by studying students' motivation towards programming at Al-Imam University to see how that affects their performance. As previous work shows the benefits of a CS0 course prior to the CS1 course, we also want to see how marks in preparatory year courses might allow better guidance for students and whether these marks can be used as predictors for student success in CS1. In addition, we plan to see if there is gender variation in programming course marks and variations as all students in this study were male.

Acknowledgements

This work would not have been successful without help from the CS1 instructors (Dr. Almohimeed, Dr. Zouhir, Dr. Mandour, Dr. Yassine, and Dr. Mohamed-Foued) in the Department of Computer Science at Al-Imam University, who provided marks for analysis. A special thanks to the course coordinator Dr. Tezeghdanti.

References

Benford, R., Gess-Newsome, J. (2006). *Factors Affecting Student Academic Success in Gateway Courses at Northern Arizona University.* 152.

Bergin, S., Reilly, R. (2005). Programming: factors that influence success. In: *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*. New York, NY, USA: ACM, 411–415. DOI: 10.1145/1047344.1047480

Brown, M. (2013). CS0 As an indicator of student risk for failure to complete a degree in computing. *J. Comput. Sci. Coll.*, 28(5), 9–16. Retrieved from: http://dl.acm.org/citation.cfm?id=2458569.2458572

Butcher, D.F., Muth, W.A. (1985). Predicting performance in an introductory computer science course. *Communications of the ACM*. DOI: 10.1145/3166.3167

Campbell, P.F. (1984). The effect of a preliminary programming and problem solving course on performance in a traditional programming course for computer science majors. In: *Proceedings of the Fifteenth*

SIGCSE Technical Symposium on Computer Science Education. New York, NY, USA: ACM. 56–64. DOI: 10.1145/800039.808623

Chase, J.D., Okie, E.G. (2000). Combining cooperative learning and peer instruction in introductory computer science. *ACM SIGCSE Bulletin*, 32, 372–376. DOI: 10.1145/331795.331888

Corney, M., Teague, D., Thomas, R.N. (2010). Engaging students in programming. In: *Twelfth Australasian Conference on Computing Education – Volume* 103. 63–72.

Crouch, C.H., Mazur, E. (2001). Peer Instruction: Ten years of experience and results. *American Journal of Physics*, 69, 970–977. DOI: 10.1119/1.1374249

Denny, P., Luxton-Reilly, A., Hamer, J., Dahlstrom, D.B., Purchase, H.C. (2010). Self-predicted and actual performance in an Introductory Programming Course. In: *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education*. New York, NY, USA: ACM. 118–122. DOI: 10.1145/1822090.1822124

Farrell, C. (2006). Predicting (and creating) success in CS1. *Issues in Information Systems*, VII(1).

Felder, R.M., Spurlin, J. (2005). Applications, reliability and validity of the index of learning styles. *International Journal of Engineering Education*, 21, 103 – 112. DOI: 0949/-149X/91

Felder, R., Silverman, L. (1988). Learning and teaching styles in engineering education. *Engineering Education*, 78, 674–681. DOI: 10.1109/FIE.2008.4720326

Hanks, B. (2005). Student performance in CS1 with distributed pair programming. *ACM SIGCSE Bulletin*. DOI: 10.1145/1151954.1067532

Hoda, R., Andreae, P. (2014). It's not them, It's us! Why computer science fails to impress many first years. In: *16th Australasian Computing Education Conference.* 159–162.

Hu, M., Winikoff, M., Cranefield, S. (n.d.). Teaching Novice Programming using goals and plans in a visual notation. In: *the Fourteenth Australasian Computing Education Conference (ACE2012)*. Melbourne, Australia.

Keen, K.J., Etzkorn, L. (2009). Predicting students' grades in Computer Science Courses based on complexity measures of teacher's lecture notes. *Journal of Computing Sciences in Colleges*, 24, 44–48.

Konvalina, J., Wileman, S.A., Stephens, L.J. (1983). Math proficiency: a key to success for computer science students. *Communications of the ACM*. DOI: 10.1145/69586.358140

Kurtz, B.L. (1980). Investigating the relationship between the development of abstract reasoning and performance in an introductory programming class. *SIGCSE Bull.*, 12(1), 110–117. DOI: 10.1145/953032.804622

Mcdowell, C., Werner, L., Bullock, H.E., Fernald, J. (2003). The impact of pair programming on student performance, perception and persistence. In: *25th International Conference on Software Engineering, 2003. Proceedings.* DOI: 10.1109/ICSE.2003.1201243

McGill, T.J., Volet, S.E., Hobbs, V.J. (1997). Studying computer programming externally: who succeeds? *Distance Education*. DOI: 10.1080/0158791970180205

Nikula, U., Gotel, O., Kasurinen, J. (2011). A motivation guided holistic rehabilitation of the first programming course. *ACM Transactions on Computing Education*. DOI: 10.1145/2048931.2048935

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., … Paterson, J. (2007). A survey of literature on the teaching of introductory programming. *ACM SIGCSE Bulletin*. DOI: 10.1145/1345375.1345441

Pillay, N., Jugoo, V.R. (2005). An investigation into student characteristics affecting novice programming performance. *ACM SIGCSE Bulletin*. DOI: 10.1145/1113847.1113888

Porter, L., Bailey-lee, C., Simon, B. (2013). Halving fail rates using peer instruction: a study of four computer science courses. In: *SIGCSE'13: Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, 177–182. DOI: 10.1145/2445196.2445250

Porter, L., Zingaro, D. (2014). Importance of early performance in CS1: two conflicting assessment stories. In: *Proceedings of the 45th ACM technical symposium on Computer science education – SIGCSE'14.* 295–300. DOI: 10.1145/2538862.2538912

Robins, A. (2010). Learning edge momentum: a new account of outcomes in CS1. *Computer Science Education*. DOI: 10.1080/08993401003612167

Simon, B., Kohanfars, M., Lee, J., Tamayo, K., Cutts, Q. (2010). Experience report: peer instruction in introductory computing. In: *Proceedings of the 41st ACM technical symposium on Computer science education.* 341–345. DOI: 10.1145/1734263.1734381

Simon, B., Parris, J., Spacco, J. (2013). How we teach impacts student learning: Peer Instruction vs. Lecture in CS0 Jaime. In: *Proceeding of the 44th ACM Technical Symposium on Computer Science Education – SIGCSE'13*. 41. DOI: 10.1145/2445196.2445215

Tanner, H., Jones, S. (2000). Scaffolding for success: reflective discourse and the effective teaching of mathematical thinking skills. *Research in Mathematics Education*. DOI: 10.1080/14794800008520065

Teague, D., Roe, P. (2008). Collaborative Learning – towards a solution for novice programmers. In: *Tenth Australian Computing Education Conference (ACE20080)*. 147–154.

Terry R. Hostetler. (1983). Predicting student success in an introductory programming course. *ACM SIGCSE Bulletin*, 15(3).

Thomas, L., Ratcliffe, M., Woodbury, J., Jarman, E. (2002). Learning styles and performance in the introductory programming sequence. *ACM SIGCSE Bulletin*. DOI: 10.1145/563517.563352

Vihavainen, A., Airaksinen, J., Watson, C. (2014). A systematic review of approaches for teaching introductory programming and their influence on success. In: *Proceedings of the Tenth Annual Conference on International Computing Education Research*. New York, NY, USA: ACM, 19–26. DOI: 10.1145/2632320.2632349

Watson, C., Li, F. W.B., Godwin, J.L. (2014). No tests required: comparing traditional and dynamic predictors of programming success. In: *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*. New York, NY, USA: ACM, 469–474. DOI: 10.1145/2538862.2538930

Werth, L.H. (1986). Predicting student performance in a beginning computer science class. *ACM SIGCSE Bulletin*. DOI: 10.1145/953055.5701

Wiedenbeck, S., LaBelle, D., Kain, V. (2004). Factors affecting course outcomes in introductory programming. In: *Proceedings of the 16th Workshop of the Psychology of Programming Interest Group*. 97–110. Retrieved from: http://www.ppig.org/papers/16th-wiedenbeck.pdf

Wilson, B.C. (2002). A study of factors promoting success in computer science including gender differences. *Computer Science Education*, 12, 141–164. DOI: 10.1076/csed.12.1.141.8211

Wood, K., Parsons, D., Gasson, J., Haden, P. (2013). It's never too early: pair programming in CS1. In: *15th Australasian Computing Education Conference.* 13–21.

R. Alturki holds Bachelor degree in Computer Science from King Saud University, Master in Computing & Software Technology and PhD from Swansea University. He works as Assistant Professor in Computer Science at Al-Imam Mohammad Ibn Saud Islamic University. His Research Interest includes multimedia delivery over Ad hoc networks, Teaching programming and Human Computer Interaction (HCI). He is a member of the Association for Computing Machinery (ACM). He is fluent in Arabic and English.