3-22-2019

Factors Affecting Student Performance in Intermediate Programming Courses: A Mixed Method Study

Ning Yang
The University of Alabama, tmnabity@crimson.ua.edu

Teagen Nabity
The University of Alabama, tmnabity@crimson.ua.edu

FACTORS AFFECTING STUDENT PERFORMANCE IN INTERMEDIATE PROGRAMMING COURSES: A MIXED METHOD STUDY

Ning Yang
The University of Alabama
nyang@crimson.ua.edu

Teagen Nabity
The University of Alabama
tmnabity@crimson.ua.edu

ABSTRACT

There is increasing demand for computing professionals in the United States and too few college graduates to fill the projected need. While extant research has explored a wide variety of student and course attributes to predict student success in introductory programming courses, similar studies focused on intermediate and advanced courses is lacking. Our study aims to explore and identify student and course attributes impacting student performance in intermediate programming courses. Using mixed methods, we provide insights for instructors and instructional designers in developing courses to facilitate the successful completion of course learning objectives for more students, leading to more students who graduate and are able to fill IT-related roles in the future.

Keywords

Intermediate programming course, mixed methods, education, student performance

INTRODUCTION

There is a growing need for talented computing professionals in the United States. It is estimated there are between 43,000 and 60,000 computer science graduates annually (Sattar, 2018; Software Guild, 2017). Projections indicate there will be one million more computing jobs than graduates to fill them by 2020 (Software Guild, 2017); this gap is set to widen as there are expected to be 1.3 million openings for IT professionals by 2026 (Sattar, 2018). Of the six industries expected to grow the most through 2020, the technology industry pays the highest and all salaries are expected to increase by 15% or more through 2022 (Computer Science Zone, n.d.). So why are there not more graduates?

One reason is students aren't finishing degree programs in computing related fields. More than 50% of all students who start in a science, technology, engineering, or math (STEM) degree program change majors (Software Guild, 2017). Between 2003 and 2009, 59% of STEM students changing majors had been enrolled in computer science programs (Computer Science Zone, n.d.). This suggests there is potentially a problem with the design of computer science courses, the ability of some students to perform well enough to feel motivated to continue in the degree program, or some combination of the two.

The literature is rich in research on student and course attributes contributing to successful completion in introductory programming courses (Bergin and Reilly, 2005; Byrne and Lyons, 2001; Lopez et al., 2008; Simon et al., 2006; Wiedenbeck et al., 2004; Wilson and Shrock 2001). However, we have struggled to find research examining attributes leading to successfully completing intermediate or advanced coursework. Although it can be argued the first course presents the greatest hurdle for students, it can also be argued that success in intermediate programming courses is equally important as the skills developed therein provide the foundation for more complex software development. Given the general objective of degree programs is to train students for entry level positions, we believe there is value in understanding the factors contributing to student success in intermediate programming courses. Our research questions are thus:

1. Which student behaviors and attributes predict their success or failure in intermediate programming courses?
2. Which instructional decisions impact the success or failure of students in intermediate programming courses?

We study these questions using a mixed methods design, combining survey and interviews, with undergraduate information systems (IS) students enrolled in the second programming course at a university in the southeastern United States. Our aim is to identify student and course attributes impacting student performance in the intermediate course. Instructors will benefit from better understanding how to design and implement courses that facilitate successful completion of learning outcomes for more students. This may also benefit employers seeking to identify employees to train in software development. Additionally, we provide a foundation for further research on continued success in programming education.

The remainder of this paper is structured as follows: first, we review the literature on key student and course success factors for introductory programming courses. Then we present our conceptual framework, followed by a discussion of our methodology. Finally, we discuss our anticipated implications and limitations.

LITERATURE REVIEW

In designing an introductory programming course, the research is highly supportive of using a completion strategy, which emphasizes hand-tracing and understanding completed code early in the course, and provides code that is progressively more incomplete for students to generate the missing portions as the semester progresses (Carbone et al., 2000; Lopez et al., 2008; Van Merrienboer, 1990; Van Merrienboer et al., 1992; Van Merrienboer and Krammer, 1987). The alternative, generation strategy—which requires students to generate and begin understanding code simultaneously—often led to more women dropping the course and students to rate subsequent assignments as more difficult when compared to the completion strategy (Van Merrienboer, 1990). The completion strategy offers a range of benefits—including promoting metacognitive skills (Carbone et al., 2002), developing program-design and problem-solving skills (Carbone et al., 2000, 2002; Caspersen and Bennedsen, 2007; Van Merrienboer and Krammer, 1987), and encouraging bootstrapping approaches (Carbone et al., 2000; Van Merrienboer et al., 1992); however, it is unclear if these benefits apply to more advanced courses. As such, we will detail the structure of the course and its assignments in terms of the completion and generation strategies; we use the qualitative analysis to determine if these factors were perceived to affect the students' ability to succeed.

The extant literature predominantly focuses on a wide range of student-based factors to predict and explain why some students succeed in introductory programming courses and others do not. Some of these factors are characteristics based on the student's disposition and background while others are behaviors exhibited by the student. The majority of the previous studies commonly use the student's score for a specific task or their final grade for the course as the outcome of interest. One study includes instructor interviews to help identify reasons for poor performance (Carbone et al., 2009). The remainder of this section identifies some important and frequently examined factors related to student success from previous studies that are worth exploring in the intermediate programming course setting.

Prior Programming Experience

The first and most important predictor is one's prior programming or computer experience. Prior computer experience, especially prior programming experience, has been used to predict programming success in many studies (Bergin and Reilly, 2005; Byrne and Lyons, 2001; Evans and Simkin, 1989; Hagan and Markham, 2000; Holden and Weeden, 2003; Wiedenbeck et al., 2004; Wilson and Shrock, 2001). However, prior computer experience is a broad term and includes experiences of computer applications, emailing, game playing, and surfing on the web (Bergin and Reilly, 2005). Not all previous computer experience is equally influential. Wilson and Shrock (2001) found that previous programming experience in a formal class had a significant, positive impact on students' programming performances, but experience in playing computer games negatively impacted student performance. We focus specifically on past programming experience—including and beyond the introductory course—as a predictor of performance.

Problem-Solving Ability

Another predictor of success in programming courses is problem-solving ability. Problem solving involves two constituent processes: (1) identifying problems and (2) formulating systematic strategies for attempting a solution (Carbone et al., 2009; de Raadt et al., 2006; Shute, 1991; Sweedyk et al., 2005). Programming requires developers to represent and interpret abstract problems. The result of these cognitive activities is expressed in the form of logic structures and translated in a standardized formal language. Collectively, these activities are the foundation for programming. High-quality and successful code requires careful planning and logic (Wiedenbeck et al., 2004). As courses become more advanced, this skill is increasingly important. Thus, we include problem-solving ability as a second predictor of performance.

Programming Self-Efficacy

Self-Efficacy Theory suggests an individual's performance influences their self-efficacy and, therefore, can affect their future performance (Askar & Davenport, 2009). To this point, Ramalingam et al. (2004) found a significant relationship between programming self-efficacy and previous programming experience. Hence, a student's previous programming experience impacts his or her programming self-efficacy, which affects his or her future performance (Wiedenbeck et al., 2004). We believe this will hold true for additional programming courses, so programming self-efficacy is our third predictor of performance.

Attributing Success to Luck and Response toward Programming Errors and Failures

According to Attribution Theory, individuals attribute success or failure to internal of external causes. Reactions to success or failure vary depending on the attribution (Kelley, 1967). If someone attributes their failures to lack of effort (internal), they are likely to put forth more effort to obtain their goals in the future. If someone attributes their failures to bad luck (external), they are more likely to blame others for failures in the future. In addition to attribution, the response to failure or errors is also a key predictor of performance (Carbone et al., 2009; Perkins et al., 1986; Wilson and Shrock, 2001). Carbone et al. (2009) found some students, when faced with programming errors, either stopped and moved onto the next problem or constantly tried new solutions without thinking or reflecting on the cause of the error. These two responses were indicative of students who were less likely to be successful in programming (Carbone et al. 2009). Failure and errors are part of learning programming, so student attributions to luck and responses toward programming failures are likely to be predictive of performance, especially in advanced coursework.

Math (and Science) Scores

Considering the analytic and algorithmic requirements of programming, math and science ability are significant predictors of a student's success in the introductory programming courses (Askar and Davenport, 2009; Bergin and Reilly, 2005; Byrne and Lyons, 2001; Wilson and Shrock, 2001). Mathematical ability and aptitude in science are strongly correlated with programming ability and affect the initial perception of programming self-efficacy (Askar and Davenport, 2009). Unlike other factors, math and science ability are at the presage stage of learning and are not easily altered (de Raadt et al., 2005). Thus, we include math and science scores as predictors of performance.

CONCEPTUAL FRAMEWORK

The goal of our study is to identify factors affecting student performance in intermediate programming courses. In the extant research, student performance in introductory programming courses was captured by either a grade on a specific programming task or a final course grade. We use the final course grade, represented as a percentage, as the measure of student performance.

There are a variety of predictors arising from the research on introductory programming courses, a handful of which we have previously discussed. Following Social Cognitive Theory, we group our predictors into three categories: environmental, personal, and behavioral (Bandura, 2012; Keith et al., 2015). We classify the predictors affecting student performance based on prior research.

Factors classified as environmental include comfort level, assignment type (individual or group work), and family's computer experience. Comfort level measures student perceptions of the difficulty of the course, module, and assignments, as well as their anxiety level while working on the assignments (Bergin and Reilly, 2005; Wilson and Shrock, 2001). There are mixed findings for the effects of assignment type. Bennedsen and Caspersen (2005) found that teamwork is not a significant predictor of student success. Conversely, McDowell et al. (2003) found that paired work enhances the quality of students' individual programming skill. Students achieve good results even when working on their own after completing a paired assignment. Additionally, having a mother or siblings with prior computing experience was found to significantly impact students' programming self-efficacy, leading to better programming performance (Askar and Davenport, 2009).

Predictors classified as personal factors include prior programming experience, problem-solving ability, programming self-efficacy, and math and science scores. This group of factors are associated with student dispositions and characteristics that are stable or slow to change. As discussed in the previous section, these factors are well researched as relating to performance in introductory courses and we believe they are still relevant predictors of performance in intermediate programming courses.

Behavioral factors include attribution to luck, response towards programming errors and failures, time management, and motivation. This group of factors, relating to student behaviors, can be affected by teacher interventions and instructional design. Attribution to luck and response towards programming errors and failures were discussed in the previous section. In addition to these predictors, research has also found that time management is an important skill for student success. This skill was found to directly impact motivation in that a lack of time management skills reduced motivation to achieve (Carbone et al., 2009). Motivation is unstable as it is dependent on other factors, which is why we classify it as behavioral instead of personal. Prior studies indicate students with intrinsic motivation are more likely to achieve success in introductory programming courses, but a lack of programming skill negatively impacts motivation (Carbone et al., 2009). Carbone et al. (2009) found instructors can promote intrinsic motivation by helping students experience successes and reducing the focus on grades—an extrinsic motivator.

Because most of the existing literature focuses on student performance in introductory programming courses, we know very little about predictors of student success in intermediate programming courses and how students in these courses are different. With little information available in the literature, we adopt the mixed method design in this study. We concurrently collect data via survey and student interviews in an intermediate programming course. Our design is discussed in more detail next.

METHOD

We apply a concurrent mixed method design for our study. Mixed method designs combine the advantages of both qualitative and quantitative research (Creswell and Plano Clark, 2017). Our study strives to determine which factors—grouped into three categories adopted from Social Cognitive Theory (environmental, personal, and behavioral)—impact student performance in intermediate programming courses.

We collect survey data at three intervals from students enrolled in the second programming course of an undergraduate IS program at a southeastern university in the United States. Throughout the semester, we periodically interview select students about their experiences in, perceptions of, and attitudes toward the course, its structure, and assignments. This data collection is complementary in that we receive a holistic view of the predictors on performance from the surveys, and the interviews provide rich detail and insight into the student experience. Additionally, while the survey is largely based on prior research focused on success factors in introductory programming courses, the interviews may illuminate other factors unique to performance in an intermediate course. Each of these phases of data collection is discussed in turn.

The IS program at our focal institution is known for graduating students who go into a wide range of IT professions—including software development. It also boasts a project-oriented Capstone sequence with internal and external clients that often requires strong programming skills; as such, it is important students perform well in the second programming course to ensure success later in the program. Each semester, this course has more than 80 enrolled students. Although the university is in the southeast, it has a large out-of-state student population that is reflected within the IS program, providing some diversity in geography and past experiences among the students.

Quantitative Study: Survey

Survey data is collected in three waves: at the start, middle, and near the end of the semester. Table 1 lists the source for each measure. The survey at the start of the term measures students' programming self-efficacy, comfort level, problem-solving ability, demographic information, past programming experience, family's computing experience, motivation, and math and science scores in previous classes and on standardized tests. Programming self-efficacy, comfort level, problem-solving ability, and motivation are also measured at midterm and near the end of the semester; only these are measured repeatedly because they are expected to change for most students over the duration of the course. Our dependent variable (final grade) is collected from the professor with the permission of the students.

Measure	Source
Programming-Self-Efficacy	Ramalingam and Wiedenbeck, 1998; Wiedenbeck et al., 2004
Comfort Level	Wilson and Shrock, 2001
Problem-Solving Ability	Heppner and Petersen, 1982
Past Programming Experience	Bergin and Reilly, 2005
Motivation	Carbone et al., 2009

Table 1. Sources for Survey Instruments

Qualitative Study: Interviews

Of the approximately 80 students enrolled in the intermediate course, we interview between 9 and 12 students at least three times throughout the semester to capture more details about their experience. We use stratified sampling based on self-reported final grades in the introductory programming course gathered from the first round of survey collection. Three to four students are selected from those who earned As, Bs, and Cs; this guarantees variance in their performance baseline which we predict also indicates variance in their initial levels of programming self-efficacy.

Interviews are semi-structured and focus on perceptions of difficulty, structure of the course, assistance sought from others (i.e. classmates, teaching assistants), attributions for successes or failures, response towards programming errors, and perceptions of time management skill. Interviewees are also asked to list and rank factors they believe are important to their success in the course. After developing our interview protocol and question list, we request feedback from IS experts and instructors to improve the probability the interviews result in valuable insights.

ANTICIPATED IMPLICATIONS AND LIMITATIONS

This is, to our knowledge, the first paper using a mixed method design to study student performance in intermediate programming courses. It will help IS researchers and educators understand which factors influence student performance in intermediate programming courses and identify possible instructional design elements that facilitate improved student learning. Researchers have studied students in introductory programming courses for more than three decades. However, few studies are devoted to investigating students in advanced programming courses. If we are to encourage students to stay in computer science-related degree programs, we need to give attention to student success in later courses as well as in introductory courses. Our study starts to bridge this gap by considering students in an intermediate programming course.

We foresee two practical implications our of research. First, instructors and instructional designers will better understand the factors leading to continued student success in learning programming. Course design has the most influence on environmental predictors, though instructors can also influence behavioral predictors—such as motivation. This should result in improved course design and more students achieving learning outcomes. Second, we facilitate future research on student success in post-introductory programming courses by identifying some of the factors impacting student performance and thereby identifying where these students may differ from those in introductory courses.

The primary limitation to our study is the collection of data from a single university. Although the university—and its information systems major—includes students from across the United States, it is limited in generalizability to other universities and computing degree programs. However, our goal with this study is to explore possible student success factors in a new context; future research will be conducted with the aim of improving generalizability. Secondly, we have limited the number of predictors we are considering at this stage, so it is possible that other predictors identified in the introductory programming course literature may be (more) relevant at the intermediate level. Although the qualitative results may capture some factors not included on the survey, future research can explore additional factors. Lastly, we rely on qualitative data to identify course attributes impacting student performance rather than collecting data from a course designed using the completion strategy and one designed using the generation strategy; future research will correct this limitation to determine if the course structure truly affects performance at the intermediate level.

REFERENCES

1. Askar, P., and Davenport, D. 2009. “An Investigation of Factors Related to Self-Efficacy for Java Programming among Engineering Students,” *The Turkish Online Journal of Educational Technology* (8:1), pp. 1303–6521.
2. Bandura, A. 2012. “Social Cognitive Theory,” in *Handbook of Theories of Social Psychology., Vol. 1.*, P. A. M. Van Lange, A. W. Kruglanski, and E. T. Higgins (eds.), Thousand Oaks, CA: Sage Publications Ltd, pp. 349–373.
3. Bennedsen, J., and Caspersen, M. E. 2005. “An Investigation of Potential Success Factors for an Introductory Model-Driven Programming Course,” in *Proceedings of the First International Workshop on Computing Education Research*, ICER ’05, New York, NY, USA: ACM, pp. 155–163.
4. Bergin, S., and Reilly, R. 2005. “Programming: Factors That Influence Success,” *ACM SIGCSE Bulletin* (37:1), pp. 411–415.
5. Byrne, P., and Lyons, G. 2001. “The Effect of Student Attributes on Success in Programming,” *ACM SIGCSE Bulletin* (33:3), pp. 49–52.
6. Carbone, A., Hurst, J., Mitchell, I., and Gunstone, D. 2000. “Principles for Designing Programming Exercises to Minimise Poor Learning Behaviours in Students,” in *Proceedings of the Australasian Conference on Computing Education*, Melbourne: ACM, pp. 26–33.
7. Carbone, A., Hurst, J., Mitchell, I., and Gunstone, D. 2009. “An Exploration of Internal Factors Influencing Student Learning of Programming,” in *Proceedings of the Eleventh Australasian Conference on Computing Education*, Wellington, New Zealand, pp. 25–34.
8. Carbone, A., Mitchell, I., Gunstone, D., and Hurst, J. 2002. “Designing Programming Tasks to Elicit Self-Management Metacognitive Behaviour,” in *Proceedings of the International Conference on Computers in Education*, Auckland, New Zealand: IEEE, pp. 533–534.
9. Caspersen, M. E., and Bennedsen, J. 2007. “Instructional Design of a Programming Course − A Learning Theoretic Approach,” in *Proceedings of the Third International Workshop on Computing Education Research*, pp. 111–122.
10. Computer Science Zone. (n.d.). *The Technology Job Gap*. (https://www.computersciencezone.org/technology-job-gap/).
11. Creswell, J. W., and Plano Clark, V. L. 2017. *Designing and Conducting Mixed Methods Research*, (3rd ed.), SAGE Publications.
12. Evans, G. E., and Simkin, M. G. 1989. “What Best Predicts Computer Proficiency?,” *Communications of the ACM* (32:11), pp. 1322–1327.
13. Hagan, D., and Markham, S. 2000. “Does It Help to Have Some Programming Experience before Beginning a Computing Degree Program?,” *SIGCSE Bulletin* (32:3), United States: ACM ASSOCIATION FOR COMPUTING MACHINERY, p. 25.
14. Heppner, P. P., and Petersen, C. H. 1982. “The Development and Implications of a Personal Problem-Solving Inventory,” *Journal of Counseling Psychology* (Vol. 29).

15. Holden, E., and Weeden, E. (2003) "The Impact of Prior Experience in an Information Technology Programming Course Sequence," in *Proceedings of the 4th Conference on Information Technology Curriculum*, CITC4 '03, New York, NY, USA: ACM, 41–46.
16. Keith, M. J., Babb, J. S., Lowry, P. B., Furner, C. P., and Abdullat, A. (2015) "The Role of Mobile-Computing Self-Efficacy in Consumer Information Disclosure," *Information Systems Journal*, 25, 637–667.
17. Kelley, H. H. (1967) "Attribution Theory in Social Psychology," *Nebraska Symposium on Motivation,* 15, 192–238.
18. Lopez, M., Whalley, J., Robbins, P., and Lister, R. (2008) "Relationships between Reading, Tracing and Writing Skills in Introductory Programming," in *Proceedings of the Fourth International Workshop on Computing Education Research*, Sydney, Australia, 101–112.
19. McDowell, C., Werner, L., Bullock, H. E., and Fernald, J. (2003) "The Impact of Pair Programming on Student Performance, Perception and Persistence," in *Proceedings of the 25th International Conference on Software Engineering*, Portland, OR, USA, January, 602–607.
20. Van Merrienboer, J. J. G. (1990) "Strategies for Programming Instruction in High School: Program Completion vs. Program Generation," *Journal of Educational Computing Research*, 6, 3, 265–285.
21. Van Merrienboer, J. J. G., Jelsma, O., and Paas, F. G. W. C. (1992) "Training for Reflective Expertise: A Four-Component Instructional Design Model for Complex Cognitive Skills," *Educational Technology Research and Development,* 40, 2, 23–43.
22. Van Merrienboer, J. J. G., and Krammer, H. P. M. (1987) "Instructional Strategies and Tactics for the Design of Introductory Computer Programming Courses in High School," *Instructional Science*, 16, 3, 251–285.
23. Perkins, D. N., Hancock, C., Hobbs, R., Martin, F., and Simmons, R. (1986) "Conditions of Learning in Novice Programmers," *Journal of Education Computing Research*, 2, 1, 37–55.
24. de Raadt, M., Hamilton, M., Lister, R., Tutty, J., Baker, B., Box, I., Cutts, Q., Fincher, S., Harmer, J., Haden, P., Petre, M., Robins, A., Simon, Sutton, K., and Tolhurst, D. (2005) "Approaches to Learning in Computer Programming Students and Their Effect on Success," in *Proceedings of the 28th HERDSA Annual Conference*, Sydney: Higher Education Research and Development Society of Australasia, Inc, 407–414.
25. de Raadt, M., Watson, R., and Toleman, M. (2006) "Chick Sexing and Novice Programmers: Explicit Instruction of Problem Solving Strategies," in *Proceedings of the 8th Australasian Conference on Computing Education*, ACE '06, Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 55–62.
26. Ramalingam, V., LaBelle, D., and Wiedenbeck, S. (2004) "Self-Efficacy and Mental Models in Learning to Program," *SIGCSE Bulletin,* 36, 3, 171.
27. Ramalingam, V., and Wiedenbeck, S. (1998) "Development and Validation of Scores on a Computer Programming Self-Efficacy Scale and Group Analyses of Novice Programmer Self-Efficacy," *Journal of Educational Computing Research,* 19, 4, 367.
28. Sattar, E. (2018) "Opinion: What Are You Going to Do about IT Skills Gap?," CIO. (https://www.cio.com/article/3269366/it-skills-training/what-are-you-going-to-do-about-it-skills-gap.html).
29. Shute, V. J. (1991) "Who Is Likely to Acquire Programming Skills?," *Journal of Educational Computing Research,* 7, 1, 1–24.
30. Simon, Fincher, S., Robins, A., Baker, B., Box, I., Cutts, Q., de Raadt, M., Haden, P., Hamer, J., Hamilton, M., Lister, R., Petre, M., Sutton, K., Tolhurst, D., and Tutty, J. (2006) "Predictors of Success in a First Programming Course," in *Proceedings of the 8th Australasian Conference on Computing Education*, 52, 189–196.
31. Software Guild. (2017) "Bridging the Software Skills Gap," SoftwareGuild.Com. (https://www.thesoftwareguild.com/blog/bridging-software-skills-gap/).
32. Sweedyk, E., DeLaet, M., Kuffner, J., and Slattery, M. C. (2005) "Computer Games and CS Education: Why and How," *SIGCSE Bulletin* 37, 1, 256.
33. Wiedenbeck, S., Labelle, D., and Kain, V. N. R. (2004) "Factors Affecting Course Outcomes in Introductory Programming," in *16th Annual Workshop of the Psychology of Programming Interest Group,* Carlow, Ireland, pp. 97–109.
34. Wilson, B. C., and Shrock, S. (2001) "Contributing to Success in an Introductory Computer Science Course: A Study of Twelve Factors," *ACM SIGCSE Bulletin,* 33, 1, 184–188.