Engineering Educators Bringing the World Together
2025 ASEE Annual Conference & Exposition
Palais des congrès de Montréal, Montréal, QC • June 22–25, 2025 ASEE
Paper ID #45621

Active Learning and Specifications Grading for Undergraduate Algorithms and Data Structures courses

Dr. Mahima Agumbe Suresh, San Jose State University

Mahima Agumbe Suresh is an Assistant Professor at San Jose State University. She received her Ph.D. from the Department of Computer Science and Engineering at Texas A&M University in December 2015. After her Ph.D., she was a postdoctoral researcher at Xerox Research Labs, India, where she worked on crime analytics and process mining. Her research interests include edge computing, machine learning, modeling and system design for cyber-physical systems and the Internet of Things, and engineering education. She has published in several peer-reviewed conferences and journals and has been a program committee member at several conferences.

Active Learning and Specifications Grading for Undergraduate Algorithms and Data Structures courses

Abstract

Algorithms and Data Structures are core concepts taught in all computing undergraduate programs. It is important to ensure that student activities in the class lay the foundation and prepare them for future courses and career. In addition, assessment should allow for students to develop a growth mindset. The course may benefit with a grading system can be designed to allow for students to revise and resubmit their programming tasks, have formative assessments to test their understanding, and focus on students mastering skills rather than chasing after points. In addition, active learning is a great way to provide hands-on engagement with the materials, which might prepare students better for future courses and career. This paper describes the author's experience in designing activities and specifications grading for an undergraduate core algorithms and data structures course.

Introduction

Algorithms and data structure design are fundamental concepts in the computing disciplines. It is listed as an element of computing knowledge[1 2] in the ACM recommended curricula. A basic understanding of core knowledge in this area is expected of students who graduate with an undergraduate degree in the computing area. With more effective engagement and pedagogical support for learning from mistakes in these courses, instructors can better prepare students to be successful in the workplace.

In the last three academic years, the courses in the author's department have had a failure rate of 7%, which amounts to 26 students. Students who are underrepresented minorities have had a GPA gap of 0.47, higher than the historic average of 0.26, and is the course with the second highest equity gap in the author's department. Similar courses in other programs in other colleges show high equity gaps. These are students who would be delayed in their 4-year graduation timelines, since this course is a prerequisite for many future courses. The problem of closing equity gaps can be handled in two ways - the way the course is taught and the way students are graded. This paper presents the author's experience in both these approaches, i.e., active learning and specifications grading, as taught in Spring 2024.

Active learning has been shown to be an effective strategy to reduce equity gaps[3] and improve students' participation and engage them to take an active role in their learning[4]. However, many classes are still taught in traditional lecture-style sessions[5].

Grading is an essential aspect of every course and is necessary to assess students' achievement of course learning objectives. Point-based systems have been known to have shortcomings[6,7], specifically, students have more motivation to accumulate points rather than meet the course learning outcomes. Specifications grading[8] is a new grading tool that makes it possible to systematically shift the focus toward student learning and developing a growth mindset. Past examples of successful implementations in STEM and other fields motivate this work[9,10,11,12,13,14].

Motivation

Active Learning is a pedagogical approach with no clear definition. However, it is widely accepted that it is an approach that allows students to think and engage with materials by doing rather than listening[15 16 17]. A meta analysis of studies that compare traditional teaching to active learning approaches shows that active learning improves exam scores and reduces failure rates[18]. However, instructors are still hesitant to adopt the approach[5]. Felder and Brent[19] have also highlighted techniques that can be easily adopted. This work is motivated by these well known techniques.

In her book on Specifications Grading[8], Linda Nilson highlights the benefits of using the system. The system has been previously implementated in several STEM courses[10,9,14]. The grading systems has the following key tenets.

1. Pass/fail grades in place of points with clear specifications
2. Revision and retakes for most assessments
3. Bundles to determine final letter grade - higher grades imply deeper or broader understanding of concepts
4. Linking grades to learning outcomes

This paper discusses how active learning can be incorporated into the grading system as well.

Active Learning

In this section, different types of class activities are described.

Example of 15-minute activity instructions

The following lists a 15-minute activity that students work on. This was from a class on recursion. The students are shown two recursive problems: one to find the nth number of the fibonacci sequence, and the other to convert a number from decimal to binary.

The instructions and prompts provided to students:

What to do: Work on the two recursive problems discussed in the class. If you have more time, pick more problems from the next slide. Alternate roles for each problem and discuss the base

case and the recursive formulation. Pick an example and convince each other that the formulation is correct.

What to submit: Submit a file/text input detailing the following:

For each problem:

- base case
- recursive formulation
- example showing that it works

Reflecting on what you discussed:

- For which problem did you do the base case and for which ones did you do the recursive formulation?
- Which of these seems easier to figure out?
- Did you try to code any of these problems? If so, was it easier to do the code after solving the problem on pen-and-paper?

If you figured out a trick based on your discussions, please mention it here.

Example full class activity - The Gift Finding Problem

This was an activity that spanned the entire 50 minutes of lecture time. Students were given a handout with instructions and prompts. Students worked in groups of two. Following the activity, the students shared out their thoughts. The instructor and students engaged in further discussions about searching algorithms, including how the problem would change if the items were sorted.

The instructions and prompts provided to students: Dudley has received only 36 gifts on his 11th birthday. He is very upset that he received fewer gifts than his previous birthday. His parents calm him down by telling him that they received their favorite racing bike.

Dudley takes help from his reluctant cousin, Harry, to use magic to find out which box contains this gift. Harry has the power to visualize the contents of a box if he can use a spell on it. Harry agrees to help Dudley but only if he is provided EXACT instructions. Dudley can only provide the following instructions:

- PW x: Point wand at box in position x
- CS: Cast spell on box
- COMP m: Compare the visualized contents to a search item m
- RES: Tell Dudley the result of the comparison
- REP n: Repeat a previous n steps
- STOP: Stop

The gift boxes are arranged in a way in which the time it takes to point a wand at each box is the same.

Prompt 1: List the set of instructions that Dudley should give Harry to find his racing bike.

Prompt 2: If the gifts are sorted in alphabetical order, would you change your instructions?

Example activity spanning multiple classes

The following is an activity that spans multiple classes. The students were provided with a basic set of instructions on creating a 2D vector class and include functionality in it using C++. This was meant to get students to refresh their knowledge of object oriented programming and C++ programming, which was required to be successful in the lab components of the course.

To convert a regular programming assignment into active learning, some components of this activity were provided as a pair programming task, where students took turns to be the main programmer. The entire task was split into short 5-10 minute programming activities, followed by 5-10 minutes of sharing out to the class. Students were asked to present their ideas. For example, they were asked if they implemented operator+ as a member function or a friend global function. They were asked to reason about their choice.

The instructions and prompts provided to students:

What to do: Create a class called Vector2D with the fields and functionalities we discussed in class. Basic functionality - include at least one attribute and one method. Play around with any 4 out of the following:

- Operator overloading:
  - operator+
  - operator-
  - operator $<<$
  - operator $>>$
  - any other operator
- Function overloading.
- In member functions, e.g., add, get magnitude
- In the main file, i.e., global scope, e.g., add
- Any other function
- Inheritance: Create a class that inherits from Vector2D and implement one other method
- In the main file, i.e., global scope, try a simple function like void print(Vector2D &) and try playing around with virtual functions
- Create another example of inheritance to play around with virtual functions and abstract classes

Grade	*Quizzes*	*Labs*	*Prog. Exam*	*Exam*	*Class Activities*
A plus	Pass RQ + 7 CQ	RL + 3LP + 5HP	A+	A+	Complete 5 CA
A	Pass RQ + 7 CQ	RL + 4LP + 4HP	A	A	Complete 5 CA
A minus	Pass RQ + 7 CQ	RL + 5LP + 3HP	A-	A-	Complete 5 CA OR (G-CA >= A AND 4 CA)
B plus	Pass RQ + 6 CQ	RL + 6LP + 2HP	B+	B+	Complete 4 CA OR (G-CA >= A AND 3 CA)
B	Pass RQ + 6 CQ	RL + 7LP + 1HP	B	B	Complete 4 CA OR (G-CA >= A- AND 3 CA)
B minus	Pass RQ + 6 CQ	RL + 8LP	B-	B-	Complete 4 CA OR (G-CA >= B+ AND 3 CA)
C plus	Pass RQ + 5 CQ	RL + 7LP	C+	C+	Complete 3 CA OR (G-CA >= B+ AND 2 CA)
C	Pass RQ + 5 CQ	RL + 7LP	C	C	Complete 3 CA OR (G-CA >= B AND 2 CA)
C minus	Pass RQ + 5 CQ	RL + 7LP	C-	C-	Complete 3 CA OR (G-CA >= B- AND 2 CA)
D plus	Pass RQ + 4 CQ	RL + 6LP	D+	D+	Complete 2 CA OR (G-CA >= B- AND 1 CA)
D	Pass RQ + 3 CQ	RL + 6LP	D	D	Complete 2 CA OR (G-CA >= C+ AND 1 CA)
D minus	Pass RQ + 2 CQ	RL + 5LP	D-	D-	Complete 2 CA OR (G-CA >= C AND 1 CA)
F	Did not meet specifications for D minus				

Figure 1: Grading bundle

For each thing you experiment with, show their usage in the main method.

What to submit: Submit your code files, and a file/text input detailing and reflecting on what you tried.

Specifications Grading

There are five categories of assessment tools in this course: Quizzes, Labs, Exams, Programming Exams, and Class Activities. The final letter grade will be determined using the specifications that follows. Note that all components of a grade must be met to achieve it.

These specifications in table 1 were designed based on the course learning objectives. Achieving these outcomes would ensure that students have the expected knowledge in algorithms and data structures corresponding to their letter grade. Details on how to meet the specifications to achieve pass/fail or high pass/low pass were provided with the corresponding assessments. Students were encouraged to ask questions to understand the grading scale.

Grade	Criteria
A+	Passed all exams. Best grade is A+.
A	Passed all exams. Best grade is A.
A-	Passed all exams. Best grade is A-.
B+	Passed all exams. Best grade is B+.
B	Passed all exams. Best grade is B.
B-	Passed all exams. Best grade is B-.
C+	Passed all exams. Best grade is C+.
C	Passed one midterm exam and final. Best grade is a B or better.
C-	Passed one midterm exam and final. Best grade is a B- or better.
D+	Passed one midterm exam and final. Best grade is a C+ or better.
D	Passed one of the exams. Best grade is a C or better.
D-	Passed one of the exams. Best grade is a C-.
F	Did not pass any exam

Figure 2: Exam Grade

RQ - Required Quizzes

RL - Required Labs

CA - Class Activities

G-CA: Grade excluding class activity category

The exam grade is determined in figure 2. Lecture exams were conducted in a traditional manner. The cutoffs for each letter grade were determined based on learning objectives mapped to each question. Occassionaly, if the questions were phrased in a confusing manner, thresholds were reduced. Overall, the cutoffs were consistent with traditional grading letter grade cutoffs.

Programming exams contained specifications similar to lab assignments (as described below), with tasks summing up to particular letter grades.

Example specifications for a lab assignment

Task 1. Padovan Sequence:

[LP] Part 1: In a file called Padovan.txt, write the pseudocode to recursively compute the nth Padovan number . A padovan number is an extension to the Fibonacci series that is defined by the relation: P(n) = P(n-2) + P(n-3). P(0)=P(1)=P(2)=1. Clearly state your base case(s).

[LP] Part 2: Implement the pseudocode in a function called unsigned int padovan(const unsigned int &n) in a file called padovan.cpp.

[LP] Part 3: Debug through your code to determine in what order the function padovan is called and with what parameters. You may choose to use cout statements or note down the steps during the debug process manually. In the file called Padovan.txt, list the function calls and returns in the order in which they were made when n is 7. E.g., when n=5, the list of function calls and returns is: 1. In padovan with n = 5 2. In padovan with n = 3 3. In padovan with n = 1 4. Returning 1 with n = 1 5. In padovan with n = 0 6. Returning 1 with n = 0 7. Returning 2 with n = 3 8. In padovan with n = 2 9. Returning 1 with n = 2 10. Returning 3 with n = 5

Criteria	Ratings	
Padovan pseudocode	Full Marks	No Marks
Padovan code	Full Marks	No Marks
Padovan debug	Full Marks	No Marks
[HP] Comments on iteration	Full Marks	No Marks
Palindrome pseudocode	Full Marks	No Marks
Palindrome code	Full Marks	No Marks
Palindrome time complexity	Full Marks	No Marks
arraylist getMin code	Full Marks	No Marks
linkedList getMin code	Full Marks	No Marks
Code compiles and runs fine	Full Marks	No Marks

Figure 3: Rubric used for lab assignment

[HP] Part 4: In the file called Padovan.txt, discuss briefly if the recursive implementation is suitable for this function as compared to an iterative implementation.

Task 2. Check if a number if a palindrome using recursion:

[LP] Part 1: In a file called palindrome.txt, write the pseudocode to recursively check if a number is a palindrome or not. A palindrome is a number that reads the same backwards and forwards. E.g., 1234321. In your implementation, you may not use a separate function to reverse the number.

[LP] Part 2: Once you have the pseudocode, write a function bool check_palindrome(const int &n) in a file called palindrome.cpp that implements the recursive algorithm you have written. Use the main function to verify the functionality of your function.

[HP] Part 3: Determine the computational complexity of your code and report the same in palindrome.txt.

Task 3. Algorithm Analysis, Thoughts and Documentation:

[HP] Extend your implementation of the arrayList template and the linkedList template from Labs 3-5 to include two functions called elemType getMinRecursive() that finds the smallest element in a list using recursion.

Specifications: All tasks have components labeled [LP] and [HP]. If you complete ALL the LP components satisfactorily, you will receive a grade of “low pass” on the lab. If you complete ALL the LP components and 2 of the 3 HP components satisfactorily, you will receive a grade of “high pass”. If you do not meet the criteria for a “low pass”, the submission will be marked as “revision needed”.

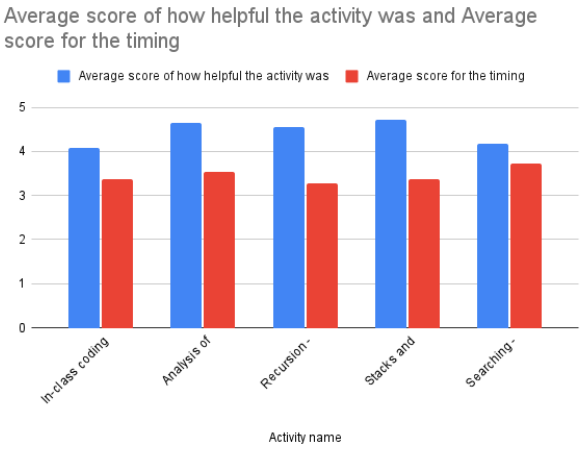

Figure 4: Class survey result for how helpful the class activities were

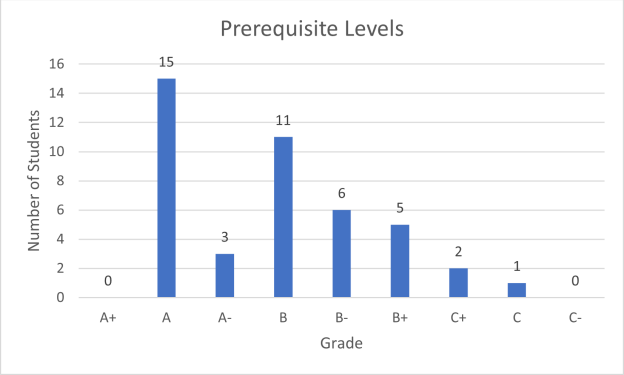

Figure 5: Prerequiste grade averages

Results

A class survey was conducted at the end of the semester and asked the following question:

Thinking back the activities we did in the class, rate each activity on a scale of 1-5. 1: The activity did not help at all 3. The activity was fine 5. The activity really helped understand the materials

The average response was 4.4.

For the following question, the average response was 3.4.

Thinking back the activities we did in the class, rate the time spent on each activity on a scale of 1-5. 1: The activity was done very fast. I wish we had more time. 3. The time spent on the activity was fine. 5. The activity took too long. We could use the time for something else.

A graph depicting the responses disaggregated by activity type is shown in figure 4.

For the feedback on the grading system, we relied on the end of semester general class survey.

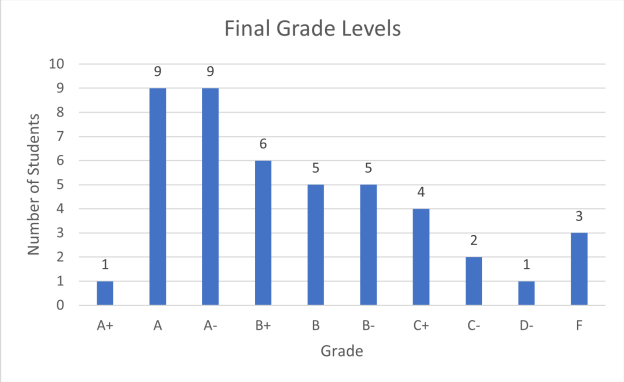

Figure 6: Final grade averages

Students did not specifically mention the grading system. This can in itself be thought of as a positive sign, since it did not cause any major issues. Students, however, mention that the revisions and resubmissions were helpful to master the material. Student performance was measured by comparing their average grades in prerequisites to their final grade in the class. The results are summarized in figures 5 and 6. Overall the number of students whose grades showed improvement outnumbered the students whose grades dropped. Some obviously noisy grade values are eliminated from this study.

Discussions

Overall, the experience was fairly successful. It motivated further tweaks to the next iteration of the system. In the next iteration, we modified the coding activity to be theme based. Students pick from a set of predefined problem spaces - e.g., library management, task management, and role playing games. They worked on this theme throughout the semester, starting with developing basic classes, operator overloading, making lists, stacks, queues, and implementing searching and sorting algorithms in the context of the problems. This also helps incorporate active learning into the specifications grading system better.

References

[1] Alison Clear and Allen S Parrish. Computing curricula 2020. *Computing Curricula*, 2020.

[2] Association for Computing Machinery (ACM) Joint Task Force on Computing Curricula and IEEE Computer Society. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. Association for Computing Machinery, New York, NY, USA, 2013. ISBN 9781450323093.

[3] Elli J Theobald, Mariah J Hill, Elisa Tran, Sweta Agrawal, E Nicole Arroyo, Shawn Behling, Nyasha Chambwe, Dianne Laboy Cintrón, Jacob D Cooper, Gideon Dunster, et al. Active learning narrows achievement gaps for underrepresented students in undergraduate science, technology, engineering, and math. *Proceedings of the National Academy of Sciences*, 117(12):6476–6483, 2020.

[4] Karl A Smith, Sheri D Sheppard, David W Johnson, and Roger T Johnson. Pedagogies of engagement: Classroom-based practices. *Journal of engineering education*, 94(1):87–101, 2005.

[5] Angélica Burbano, Katherine Ortegon, Silvia Guzman, and Henry Arley Taquez Quenguan. Active learning: Faculty mind-sets and the need for faculty development. In *2019 ASEE Annual Conference & Exposition*, 2019.

[6] Thomas R Guskey and Jane M Bailey. *Developing grading and reporting systems for student learning*. Corwin Press, 2001.

[7] Mark Carl Rom. Grading more accurately. *Journal of Political Science Education*, 7(2):208–223, 2011.

[8] Linda B Nilson. *Specifications grading: Restoring rigor, motivating students, and saving faculty time*. Stylus Publishing, LLC, 2015.

[9] Kate J McKnelly, William J Howitz, Taylor A Thane, and Renée D Link. Specifications grading at scale: Improved letter grades and grading-related interactions in a course with over 1,000 students. 2022.

[10] William J. Howitz, Kate J. McKnelly, and Renée D. Link. Developing and implementing a specifications grading system in an organic chemistry laboratory course. *Journal of Chemical Education*, 98(2):385–394, 2021. doi: 10.1021/acs.jchemed.0c00450.

[11] Dennis Earl. Two years of specifications grading in philosophy. *Teaching Philosophy*, 45(1):23–64, 2022.

[12] Ella Tuson and Tim Hickey. Mastery learning and specs grading in discrete math. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*, ITiCSE '22, page 19–25, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392013. doi: 10.1145/3502718.3524766. URL https://doi.org/10.1145/3502718.3524766.

[13] Brian P. Helmke. Specifications grading in an upper-level bme elective course. In *2019 ASEE Annual Conference & Exposition*, number 10.18260/1-2–33278, Tampa, Florida, June 2019. ASEE Conferences. https://peer.asee.org/33278.

[14] Mahima Agumbe Suresh. Challenges and experiences in implementing a specifications grading system in an upper-division undergraduate computer networks course. In *2023 ASEE Annual Conference & Exposition*, 2023.

[15] Michael Prince. Does active learning work? a review of the research. *Journal of engineering education*, 93(3): 223–231, 2004.

[16] Charles C Bonwell and James A Eison. Active learning: Creating excitement in the classroom. school of education and human development, george washington university, 1991.

[17] Jim Eison. Using active learning instructional strategies to create excitement and enhance learning. *Jurnal Pendidikantentang Strategi Pembelajaran Aktif (Active Learning) Books*, 2(1):1–10, 2010.

[18] Scott Freeman, Sarah L Eddy, Miles McDonough, Michelle K Smith, Nnadozie Okoroafor, Hannah Jordt, and Mary Pat Wenderoth. Active learning increases student performance in science, engineering, and mathematics. *Proceedings of the national academy of sciences*, 111(23):8410–8415, 2014.

[19] Richard M Felder and Rebecca Brent. Active learning: An introduction. *ASQ higher education brief*, 2(4):1–5, 2009.