2023 Annual Conference & Exposition
Baltimore Convention Center, MD | June 25 - 28, 2023
The Harbor of Engineering
Education for 130 Years
ASEE
Paper ID #37079

Incorporating a Teach for Mastery System in a Computer Science Course

Lea Wittie, Bucknell University

Lea Wittie is an Associate Professor in the department of Computer Science in the Engineering College at Bucknell University.

Incorporating a Teach for Mastery System in a Computer Science Course

Lea Wittie
Department of Computer Science
Bucknell University
Lewisburg, PA 17837
Email: lwittie@bucknell.edu

Abstract

Teaching for Mastery is the idea that students should progress through material when they have conquered the previous material it depends on and each student may progress at their own pace. The paper documents the process and results of incorporating a teach for mastery system into a Computer Science Programming Language Design course. The course is aimed at junior and senior undergraduate students and its goal is to introduce them to the four main language families and to show them how to teach themselves a programming language. This teach for mastery system caused both student learning and student satisfaction to improve in a course that was already well received before these changes. Other benefits of this system include a marked decrease in the amount of time spent grading and a straightforward way to identify which subjects need to be re-thought and which are being learned to the intended depth. The system does come with an increase in the amount of contact time spent with students which is both a positive in that you develop a closer relationship with the students and a negative in that it is a large amount of contact time. Contributions include the structural changes and teaching artifacts needed to implement teaching for mastery in a course. The structural changes include flipping the classroom and replacing traditional sit-down exams with short one-topic take-when-ready exams. The teaching artifacts produced include a concept map. Students leave the course with a clear list of which topics they mastered and which they are still working on. This model still permits room for traditional laboratory and project components.

Introduction

In standard teaching, course topics are covered on a set schedule and exams occur at set schedules. If students have not learned the material for an earlier portion of the class, the course moves along anyway and focuses on new more difficult material that builds on the previous material that student still does not understand. This promotes a fixed mindset promoting the idea that if you didn’t get a concept, you never will. At the end of the course, students may only have a partial understanding of the material and may be unclear on what they know and don’t know.

Teach for Mastery (TfM) is a concept pioneered in the 1920s[1] with slow but steady research over the decades, including work by Bloom[2], and recently made popular by Sal Khan of Khan Academy in a 2015 TED Talk[3]. The basic idea is let each student progress through the material at their own individual pace[4,5]. Students can take and re-take both learning modules and assessments as many times as it takes for them to understand the material. This promotes a growth mindset and encourages students to take time as needed because its never too late (until the course ends). TfM is gaining a foothold in higher education and research into it has largely found that it has positive effects on student learning[6,7,8].

A flipped classroom is an educational system where the traditional lecture portion of the class has been moved to readings and/or videos to be watched outside of class and the homeworks have replaced them as the in-class work. Prior work has been done on TfM in the out-of-class portion of a flipped classroom[9] and TfM to precede a project based component of a course[10]

Methods

This section describes the documents and items needed in order implement teach for mastery. These include an autograder, sets of reading questions, activities, home/in-class assignments, a large collection of exam and final exam questions, a concept map, and progress reports. The course described here teaches programming language design concepts to junior and senior students using a flipped classroom, TfM system. It also has a laboratory component and a course project. The course meets for 3 lecture hours and 2 lab hours each week.

Autograder An autograder is software that provides instant grading and feedback on student work. Its purpose is to take the burden of manual grading off the professor thus freeing up their time[11,12]. Some common options are Moodle Quiz, Moodle Coderunner[13], Blackboard Quiz, Coursersa Autograder[14], Replit[15], Gradescope[16], Codepost[17], and Autograder[18].

A code autograder runs the student code through a series of tests, some visible to the student, and some hidden. It returns a score and also feedback for all (visible) tests. The hidden tests keep students from gaming the system by printing the expected response without doing the requested calculation.

Although research has suggested using regression penalties to avoid overreliance on autograders[19], the penalties approach does not work with a TfM system so I instead suggest delays between re-attempts.

Video/reading lectures As in a standard flipped classroom[?] , the lectures are replaced by videos and/or readings which the students watch/read each day before class. Video and reading questions serve to both check students understanding and act as a mechanism to force the students to watch/read the lecture material.

The videos for this course range from 3 to 12 minutes each. They are carefully prepared presentations rather than the live lecture recordings many professors, myself included, created at the start of the COVID pandemic. They include subtitles to make them more accessible to non-native English speakers and make it possible to watch them with the sound off or at high playback speed. The video questions can be answered with information directly contained in the video. They give instant scoring and feedback with suggestions when completed and are infinitely repeatable after a 5 minute delay so that students can rewatch the video as needed to glean the necessary information.

Figure 1 shows a sample video question and its feedback for incorrect answers. The video covered the six paradigms mentioned here and the hallmarks of those paradigms and language examples of each including Java. The students are familiar with Java from a previous course.

Video Question: Java is a member of which Paradigms? Select all that apply.

○ Imperative ○ Declarative ○ Object Oriented ○ Functional ○ Logical ○ Scripting

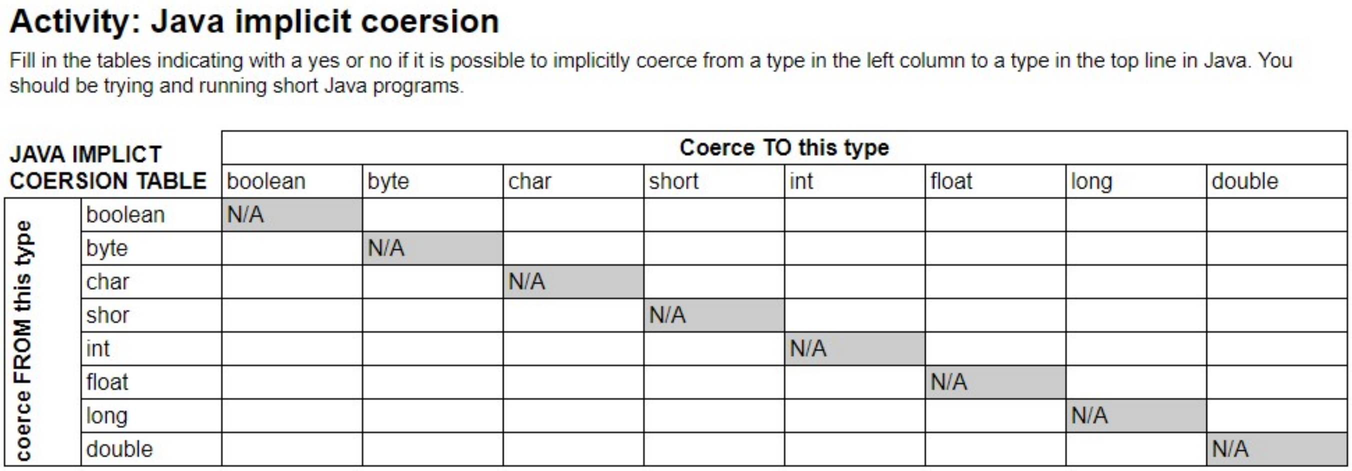
Feedback with each choice

Imperative	Correct
Declarative	Nope, look up Java code and check it out if you are unsure. Declarative languages center around logic (true and false) or around recursive functions with no loops or assignments.
Object Oriented	Correct
Functional	Nope. functional languages center around recursive functions with no loops or assignments. Look up Java code and check it out if you are unsure.
Logical	Nope, logic languages center around true and false. Look up Java code and check it out if you are unsure.
Scripting	Nope, look up Java code and check it out if you are unsure. To print out hello world, you need to write like 10 lines of code. A scripting language would do the whole hello world in about one line.

Figure 1: Sample video question and its feedback.

In-class Activities The students are given a set of one or more ungraded activities to be done in class each day. These are often communal activities where the students together contribute parts of a task using the day's topics or solo activities where each student tries to accomplish a task on their own. Either way, the students are encouraged to discuss the topic amongst each other and to ask questions.

For example, on the day when the course covers type coercions, the activity seen in Figure 2 asks the students to see how Java does implicit coercion. The students contribute to two large tables as a class. These can be done on the board if the course is in person or in a communally shared spreadsheet (such as on Google Drive) if the class is remote but synchronous. There is also a version of the activity for students who miss class. These activities are ungraded and allow the students to actively engage with the material during class time. When possible, students have access to answer keys after the activity is completed.



Activity: Java implicit coersion

Fill in the tables indicating with a yes or no if it is possible to implicitly coerce from a type in the left column to a type in the top line in Java. You should be trying and running short Java programs.

JAVA IMPLICT COERSION TABLE		Coerce TO this type							
		boolean	byte	char	short	int	float	long	double
coerce FROM this type	boolean	N/A							
	byte		N/A						
	char			N/A					
	shor				N/A				
	int					N/A			
	float						N/A		
	long							N/A	
	double								N/A

Figure 2: Sample in-class activity.

In-class Question: Below is one code program in two statically typed languages Avaj and Coffee that look a lot like Java but may have made different language design choices. Avaj and Coffee are identical except for this choice.

```
class Tree {
    void bloom() { print( "flowers bloom" ); }
}
class AppleTree inherits Tree {
    void bloom() { print( "apples appear" ); }
}
Tree tree = new AppleTree();
tree.bloom();
```

Avaj prints ”apples appear” therefore Avaj is menu ↓ method bound.

Coffee prints ”flowers bloom” therefore Coffee is menu ↓ method bound.

The menu choices are: dynamically, statically, and unknown.

Figure 3: Sample home/in-class question.

In-class work After the day’s activities are done, the students tackle a set of one or more homework-like questions, also in-class. These are solo tasks but the students are encouraged to discuss them with each other and to ask questions. These are auto-graded and students receive instant feedback of both scoring and suggestions when their answers are incorrect. Students can re-attempt these tasks an infinite number of times but there is a built in delay of several minutes between attempts in order to encourage students to ask questions of their classmates and the professor.

For example, when this language course covers static and dynamic method binding, a select-from-a-menu question seen in Figure 3 is part of the in-class work. Students work the problem out on paper or by running code examples and then select their answers. The in-class work can also include auto-graded coding assignments. Figure 4 shows in-class work on unification in Prolog. The autograder returns feedback designed to help student correct their errors.

In-class Coding:

Separate from the issue of type checking, is the issue of running a program. Prolog is not excellent at math and cannot generally do math backwards (given the output, find the inputs) or we would be using it to undo multiplication and break cryptography. Finish the following rule to do addition. Put your solution in a file named addition.pl.

add(Value1, Value2, Answer) :- % you fill in the rest

Hints

- Should you use =, is, or == to calculate the answer?
- This predicate is only meant to be used in situations where you know the first two parameters. Do not call it with variables as either of the first two parameters in the call.
- You will likely need to write more than one rule named add.
- You may need to test the type of one of the parameters to know whether to use a rule with + or a rule with string_concat.
- Do not worry about calls to add like add(3, 3.14) or add(3, “hi”) for this assignment.

Autograder test cases

test case	**fail response(s)**
add(3, 4, 7)	Doh! Your results were: **results**.
add(3.2, 4.6, 7.8)	Doh! Your results were: **results**.
add(”hi”, ” there”, ”hi there”)	Doh! Your results were: **results**.
Uses correct operator.	Doh! == checks if things are already the same. Doh! = unifies without evaluating.

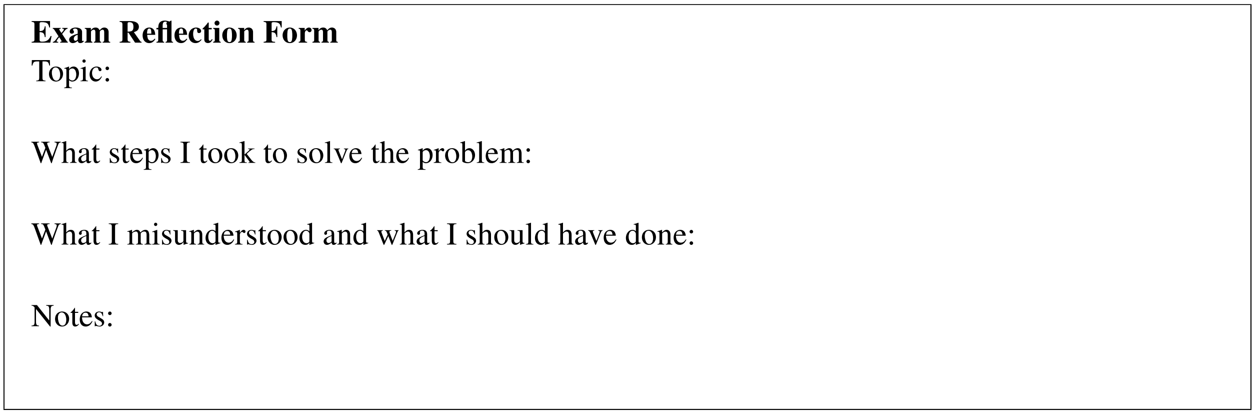
Figure 4: Sample home/in-class coding question with test cases and feedback.

Exam questions Each exam question should aim to test knowledge of just one topic. Each exam thus has on average one short question in it. This requires many versions of each question. Students can opt to take the exam on a topic at any point in class provided they have scored 90% or higher on the relevant in-class questions.

They get a random question from the set of possible questions on that topic and the questions are all auto-graded. Students receive instant feedback about their score but do not have access to the question or their answer after they submit it. Students can re-attempt these exam topics infinite times; getting a (hopefully) different question each time.

Students are free to access their notes and course materials while answering these questions and are encouraged to write and run test code as well. They are allowed to access the Internet in general, however, they are forbidden from using Internet chat sites such as stackoverflow.com or question answering sites such as Chegg, CourseHero, or ChatGPT or from communicating with other people using any medium.

Exam questions give the students 30 minutes to do problems that would typically take them 5-10 minutes on a standard exam. There is a delay of 46 hours minimum between re-attempts to avoid students instantly re-taking the exam and hoping for accidental success. The delay coincides with the amount of time between class meetings in a Monday/Wednesday/Friday course. In order to improve their understanding when faced with unsuccessful exam results, the student must meet with the professor between attempts and go over what they did incorrectly and why it was incorrect and what the correct answer was. During the review, the student fills out a form such as the one seen here in Figure 5. This form stays with the student and becomes both a study aid and a reminder to use during their re-attempt of the exam question. The student is free to write anything including the exam question on their form as there is a large pool of random exam questions for each topic (around 30 versions of each question).



Exam Reflection Form
Topic:

What steps I took to solve the problem:

What I misunderstood and what I should have done:

Notes:

Figure 5: Exam reflection tool used by students when going over incorrect exam questions.

As an example, Figure 6 shows a fill-in-the-blank question on non-strict parameter passing; pass by need.

In order to prevent unnecessary re-attempts to achieve a perfect score on questions where the student has already achieved a score of 90% or higher, all such scores are converted to 100% credit. The difference between a 90% score and 100% typically comes down to typos rather than a lack of understanding. This also limits the unnecessary exposure of excess versions of each exam question.

Final exam questions The final exam questions have the same setup as the exam questions except that they are harder and/or test the intersection of multiple topics. These provide a one to two hour window to the student for a question that would typically take 20-30 minutes on a traditional final exam. Like the exam questions, they have a re-attempt delay of 46 hours.

Concept Map A concept map for a course is a graphical representation of all topics taught in the course and which ones depend on which other ones. They are used to break the course into its individual components and to establish dependencies between topics. Figure 7 shows a portion of the concept map for this course. Research into creating[20,21] and evaluating[22] concept maps can provide guidance on creating one for other courses.

Exam Question: A program in the (non-existent) programming language Eek, whose syntax looks a lot like C. This is a good program and it runs correctly.

```
int a(int z, int y) {
    print(’a’)
    return y - z + 2
}
int b(int x) {
    print(’b’)
    return x - 1
}
int c(int w, int u) {
    print(’c’)
    return w + u
}
```

```
void main() {
    int w, x, y, z, t, u, v
    w = b(3)
    x = a(w, 3)
    y = b(x)
    z = c(1, 1)
    u = a(y, 6)
    v = u + 1
    t = a(z, v)
    print(t)
}
```

What prints with call by need? Put your answer in the provided box. Do not use any spaces or line breaks. I’m expecting to see a bunch of letters and a number like aabbcc4.

Figure 6: Sample exam question.

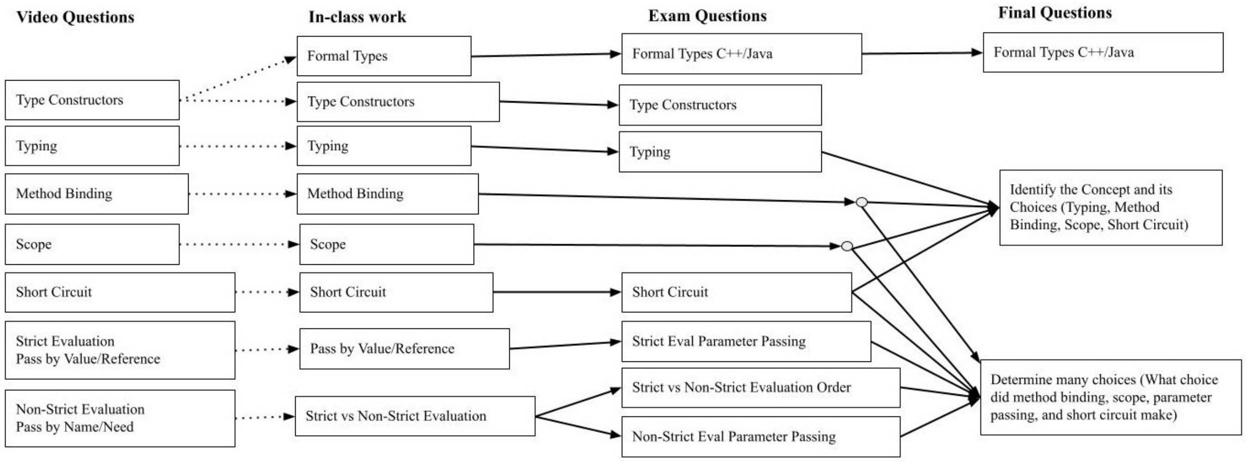
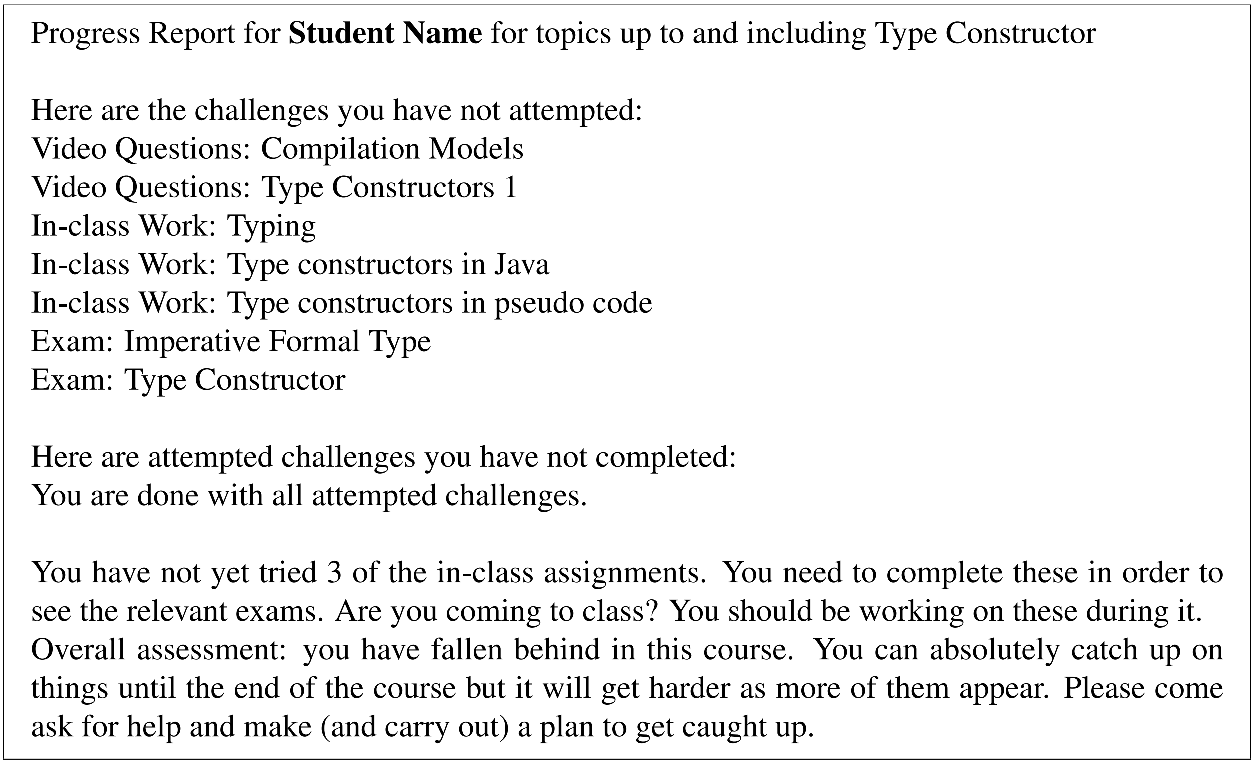

Video Questions
In-class work
Exam Questions
Final Questions
Formal Types
Formal Types C++/Java
Formal Types C++/Java
Type Constructors
Type Constructors
Type Constructors
Typing
Typing
Typing
Method Binding
Method Binding
Identify the Concept and its Choices (Typing, Method Binding, Scope, Short Circuit)
Scope
Scope
Short Circuit
Short Circuit
Short Circuit
Strict Evaluation Pass by Value/Reference
Pass by Value/Reference
Strict Eval Parameter Passing
Strict vs Non-Strict Evaluation Order
Determine many choices (What choice did method binding, scope, parameter passing, and short circuit make)
Non-Strict Evaluation Pass by Name/Need
Strict vs Non-Strict Evaluation
Non-Strict Eval Parameter Passing

Figure 7: Part of the Concept Map for the programming language course. The columns show which topics are covered by what course structures. The dependency pointers between the topics in the video questions column are omitted as the topics are taught in the order seen here from top to bottom.

Progress Reports Procrastination is a valid concern when there are not set due dates for course assignments[23,24]. One way to combat this tendency is to provide frequent reminders of work to be done. The students are sent a weekly progress report like the one seen in Figure 8 listing the assignments that they should have done but have not yet started and the ones that they are started but not yet completed. This report only includes topics that have already been covered. These reports are generated and send to the students via scripts so the process is mostly automated. By the end of the course, the progress report is a complete list of any topics the student has not sufficiently understood and can be compared with the challenge map for a list of all topics that have been completed.



Progress Report for **Student Name** for topics up to and including Type Constructor

Here are the challenges you have not attempted:
Video Questions: Compilation Models
Video Questions: Type Constructors 1
In-class Work: Typing
In-class Work: Type constructors in Java
In-class Work: Type constructors in pseudo code
Exam: Imperative Formal Type
Exam: Type Constructor

Here are attempted challenges you have not completed:
You are done with all attempted challenges.

You have not yet tried 3 of the in-class assignments. You need to complete these in order to see the relevant exams. Are you coming to class? You should be working on these during it. Overall assessment: you have fallen behind in this course. You can absolutely catch up on things until the end of the course but it will get harder as more of them appear. Please come ask for help and make (and carry out) a plan to get caught up.

Figure 8: Sample progress report for a student who is behind on several assignments.

Classroom time On a typical lecture day, the students have watched a video and completed video questions before coming to class. The instructor puts up a short reminder of the days topics to serve as a quick reference for students. The instructor guides the students through the day's activity and in-class assignment. About half way through the hour long class meeting, students begin trying exam and/or final questions. The instructor goes over incorrect exam and final questions with students one-on-one as well. As every student is likely working on a different topic/assignment/exam from the person next to them, it is a form of controlled chaos.

Results

The programming languages course had 52 students in 2021 (pre-TfM) and 47 in 2022 (with TfM). Both semesters were flipped classroom with out of class lecture videos and video questions. Both had daily in-class activities and weekly lab periods. The 2021 (pre-TfM) had homeworks, standard exams, and a standard final exam. The 2022 (with TfM) used the in-class assignments, and the one-topic exam and final assessments described in the Methods section of this paper.

Student Achievement Figure 9 compares the mean scores on exam and final topics between 2021 (pre-TfM) and 2022 (with TfM). As all scores on exams and final questions in 2022 that were 90% and above were raised to 100%, the same has been done for the 2021 scores so they can be directly compared. Several topics show a significant improvement from 2021 (pre-TfM) to 2022 (with TfM). Nine of the topics showed significant improvement, six showed no significant difference, and no topics became worse.

Topic	mean 2021 pre-TfM	mean 2022 with TfM	t-value [95% CI]	p-value
Determine Many Choices	82.843	84.906	-0.455	0.650
Identify the Concept	83	100	-4.136	0.0001
Typechecking	91.025	94.109	-0.828	0.410
Formal Type OOPL	95.0000	96.653	-0.520	0.605
Formal Type Functional	94.240	87.128	1.765	0.084
Type Constructor	84.776	94.540	-2.412	0.018
Polymorphism	61.538	100.0000	-5.6456	<.0001
Typing vs Binding	84.914	95.189	-3.586	0.0006
Prolog Unification	77.941	94.872	-3.381	0.001
Memory Hazards	76.962	88.0667	-2.910	0.005
Anonymous Functions	98.693	100.0000	-1.273	0.209
Higher Order	86.275	97.778	-2.150	0.035
Infinite Lists	61.094	83.333	-2.536	0.015
Regular Expressions	57.792	98.632	-8.259	<.0001
Precedence & Associativity	92.353	90.213	0.476	0.636

Figure 9: Results of Welch's Unequal Variance T-Test. The alternative hypothesis is that the true difference in means is not equal to 0. The alpha significance level is the standard 5%. The Determine Many CHoices topic is a final question that asks students to look over a program in a madeup language and given what it prints, determine its scoping, its method binding, whether it is pass/call by value, reference, name, or need and whether it uses short circuit or not. The Identify the Concept topic, also a final question, asks students to look over a program in an madeup language and given two options of what it could print, identify which one topic (Scoping, Typing, Method Binding, etc.) would cause the output difference and which choice went with which output.

The mean values for 2021 (pre-TfM) showed that the Polymorphism, Memory Hazards, Prolog Unification, Infinite Lists, and Regular Expressions topics (with mean scores under 80%) were the topics which needed the most improvement. Some topics covered in 2022 were not assessed in a way that could be directly compared to the assessments in 2021 so they are not shown in these results. However, the mean score on all 2022 exam and final topics was above 80% and the four 2021 topics that needed the most improvement all were significantly improved and no longer under 80% mean scores.

Skipped topics Given that the exam and final assessments in a TfM system are taken at each individual student's pace, there are likely to be a few assessments that get skipped when students fall behind or run out of time. The results shown in Figure 9 only account for students who attempted the assessment, not those who skipped it entirely. There were a few students each year who either dropped the course midway through or abruptly ceased participating and did not return. There were one such student in 2021 and two in 2022. In order to do an equal comparison of the two years, I omitted scores from these students after they ceased participating in the course. In 2021 (pre-TfM) no students skipped the scheduled exams or the final unless they had left the course. Figure 10 shows the percentage of students who skipped each exam and final assessment in 2022. Most of the skipped assessments had only 1-2 students miss them. The Formal Type questions, Prolog Unification, Determine many choices, Identify the concept, Regular Expressions, and Infinite Lists each were skipped by at least 10% of the students indicating that there is still room for improvement in those topics. Infinite Lists in particular was skipped by almost 50% of students and was not the last topic covered in the course indicating that students did not feel prepared to tackle its assessment. This is a useful indication that the course material on infinite lists needs to be re-worked.

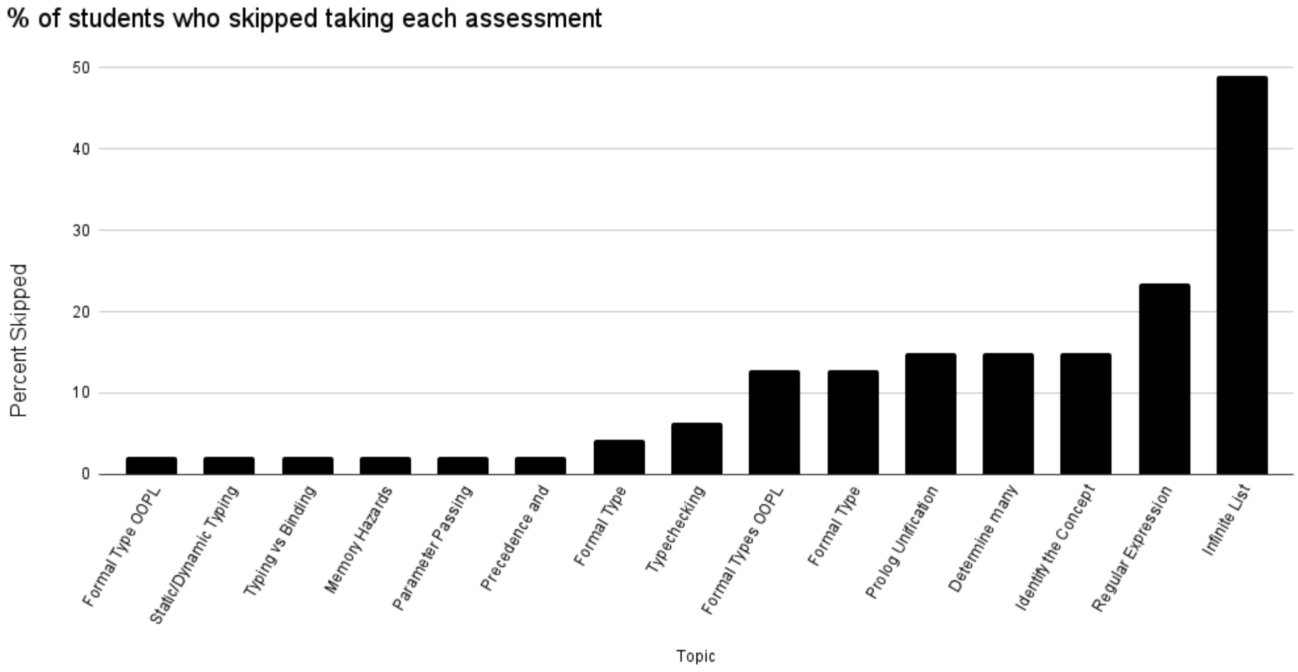


Figure 10: Percentage of students in the course who entirely skipped taking each assessment in 2022 (with TfM). Most assessments were not skipped by anyone still in the course and are not shown here. Assessments that were attempted but never reached a perfect score are also not included here.

Number of topics covered 19 topics were assessed on exams and the final in 2021 (and prior years) compared to 26 such topics in 2022. The course did not have excess unused time in 2021 or previous years. Observation of the student's efforts and the results from 2022 suggests that there were simply put too many topics in the course in 2022. The students rose to the challenge but were clearly tired of the pace by the end of the semester. In future course offerings, I plan to either reduce the number of topics to 75% of the current number or only require that students complete 75% of the available topics.

Video engagement Students frequently replayed the videos with the sound off during class to catch points they missed or did not clearly understand. They appreciated being able to play the videos at double speed and read the subtitles while they did this.

Cheating Students collaborating on exams in-class, in take home situations, using Chegg, etc has long been an issue for academia. In a TfM course, students are not motivated to collaborate on exams since they each do the topic assessments at different days and times. In order to further discourage cheating, the students were only allowed to take the exams during class, or office hours when the professor is present. Three students collaborated on an exam in 2021 (pre-TfM). No students were found cheating during 2022 (with TfM).

Exam accommodations It has become more and more common over the years for students to have accommodations such as extra time on tests. The excess time and infinite redo attempts built into this system meant that no such student requested to use their extra time accommodations in 2022 (with TfM) while they had done so routinely in 2021 (pre-TfM) and previous years.

Extra contact hours Students attempted exam and final questions 1645 times; 1082 of those were successful attempts. The remaining attempts were each followed by approximately 5 minutes of contact time with the professor to go over the question. That comes to approximately 47 hours of contact over the semester. About 20 minutes of that time was spent during each lecture each day so approximately 28 hours. The remaining 19 hours were spent during office hours. That is in addition to the usual office hours where students ask general questions about the material. With 47 students, this comes to just under 30 additional contact minutes per student over the semester. In balance however, almost no time was spent on grading assignments or assessments at all.

Student opinions Student feedback was collected though a survey (with IRB approval) which 27 of the 52 students (52%) of them took. They viewed the TfM system positively overall. Figure 11 shows the survey results. 92% of students surveyed appreciated the one-topic assessments taken when they individually are ready as opposed to standard whole-class scheduled exams and a final. In survey comments, nearly all students referred to the self-paced, one-topic, when-ready assessment structure as less stressful than traditional scheduled multi-topic assessments. Interestingly 8% (2 students) disliked this exam structure. 69% felt positively about the lack of due dates for each in-class work and exam and final question aside from the end of the semester. Another 27% of students were neutral about the due dates. This lack forced them to self-pace and to assess when they were ready to try an assessment. One student stated that their ADHD caused them difficulty in self pacing. 89% of students liked (or better) having to master each topic in order to continue on to its more difficult assessments. Unsurprisingly, 96% of students felt positively about the availability of infinite retakes on all in-class work and exam and final questions. 92% of students appreciated the weekly progress reports with a to-do list. Although one student had a neutral opinion about the overall teach for mastery system no-one voted against the system and everyone else liked it. One student even commented ”I wish more teachers would implement a system like this! so far this has been one of my favorite classes!”.

	One-topic exams	Lack of due dates	Master topic to continue	Infinite re-takes	Progress reports
Strongly Like	85%	42%	48%	89%	81%
Like	7%	27%	41%	7%	11%
Neutral	0%	27%	7%	0%	4%
Dislike	4%	4&	4%	0%	0%
Strongly Dislike	4%	0%	0%	4%	4%

Figure 11: Likert scale results for student opinions.

Conclusion The Teach for Mastery system described in this paper showed significant improvement in student achievement on a large number of topics in the course. The analysis process of examining mean topic scores and the percentage of students who skip each assessment can be done with minimal effort in a spreadsheet program and easily highlights which topics need further improvement. Students in general liked the system and found it to be less stressful than a traditional course.

References

[1] T. Nodine, “How did we get here? a brief history of competency-based higher education in the united states,” *The Journal of Competency-Based Education*, vol. 1, no. 1, pp. 5–11.

[2] B. S. Bloom, “Learning for mastery. instruction and curriculum. regional education laboratory for the carolinas and virginia, topical papers and reprints, number 1.,” *Evaluation comment*, vol. 1, no. 2, p. n2, 1968.

[3] S. Khan, “Let´ s teach for mastery, not test scores. TED talk,” 2015.

[4] R. Armacost and J. Pet-Armacost, “Using mastery-based grading to facilitate learning,” in *33rd Annual Frontiers in Education, 2003. FIE 2003.*, vol. 1, pp. T3A–20, 2003.

[5] S. Engelmann, “Student-program alignment and teaching to mastery.,” *Journal of Direct Instruction*, vol. 7, no. 1, pp. 45–66, 2007.

[6] J. M. Bekki, O. Dalrymple, and C. S. Butler, “A mastery-based learning approach for undergraduate engineering programs,” in *2012 Frontiers in Education Conference Proceedings*, pp. 1–6, 2012.

[7] B. Mostafavi, M. Eagle, and T. Barnes, “Towards data-driven mastery learning,” 2015.

[8] B. McCane, C. Ott, N. Meek, and A. Robins, “Mastery learning in introductory programming,” 2017.

[9] Y. Wang and G. Y. Qi, “Mastery-based language learning outside class: Learning support in flipped classrooms,” 2018.

[10] M. Jazayeri, “Combining mastery learning with project-based learning in a first programming course: An experience report,” in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2, pp. 315–318, 2015.

[11] K. Danutama and I. Liem, “Scalable autograder and LMS integration,” *Procedia Technology*, vol. 11, pp. 388–395, 2013. 4th International Conference on Electrical Engineering and Informatics, ICEEI 2013.

[12] P. Nordquist, "Providing accurate and timely feedback by automatically grading student programming labs," *Journal of Computing Sciences in Colleges*, vol. 23, no. 2, pp. 16–23, 2007.

[13] R. Lobb and T. Hunt, "Coderunner." https://moodle.org/plugins/qtype_coderunner, 2023.

[14] Coursera, "Coursera autograder." https://pypi.org/project/coursera-autograder/, 2023.

[15] Replit, "https://replit.com/," 2023.

[16] Gradescope, "https://www.gradescope.com/," 2023.

[17] Codepost, "https://codepost.io/," 2023.

[18] J. Peretta and M. Mior, "Autograder." autograder.io, https://github.com/eecs-autograder/autograder.io, 2023.

[19] E. Baniassad, L. Zamprogno, B. Hall, and R. Holmes, "Stop the (autograder) insanity: Regression penalties to deter autograder overreliance," in *Proceedings of the 52nd ACM technical symposium on computer science education*, pp. 1062–1068, 2021.

[20] M. Carvalho, T. C. Eskridge, and J. Lott, "Concept maps: Integrating knowledge and information visualization," *Knowledge and information visualization: Searching for synergies*, pp. 205–219, 2001.

[21] S.-S. Tseng, P.-C. Sue, J.-M. Su, J.-F. Weng, and W.-N. Tsai, "A new approach for constructing the concept map," *Computers & Education*, vol. 49, no. 3, pp. 691–707, 2007.

[22] A. J. Cañas, J. D. Novak, and P. Reiska, "How good is my concept map? am i a good cmapper?," *Knowledge Management & E-Learning*, vol. 7, no. 1, p. 6, 2015.

[23] E. Spertus and Z. Kurmas, "Mastery-based learning in undergraduate computer architecture," in *2021 ACM/IEEE Workshop on Computer Architecture Education (WCAE)*, pp. 1–7, IEEE, 2021.

[24] C. Ott, B. McCane, and N. Meek, "Mastery learning in cs1 - an invitation to procrastinate?: Reflecting on six years of mastery learning," in *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, ITiCSE '21, (New York, NY, USA), p. 18–24, Association for Computing Machinery, 2021.