INTRODUCING FLEXIBLE ASSESSMENT INTO A COMPUTER NETWORKS COURSE: A CASE STUDY

Joseph Meehean
University of Lynchburg
meehean.j@lynchburg.edu

ABSTRACT
With overall positive results and limited drawbacks, I have adapted modern pedagogical techniques to address a common difficulty encountered when teaching a computer networks course. Due to the tiered nature of the skills taught in the course, students often fail unnecessarily. Using mastery learning, competency-based education, and specifications grading as a foundation, I have developed a course that allows students with varied skills and abilities to pass. The heart of this approach is the flexible assessment of programming assignments which eliminates due dates and allows students to have their work graded and regraded without penalty. Flexible assessment also defines an interactive approach to grading which gives students immediate formative feedback and does not penalize initial failure. Using these instructional techniques, I improved the course completion rate by 30 percentage points compared to similar courses. Flexible assessment works best for upper-level courses that are not prerequisite courses because a student can pass without mastering all of the skills; their grade reflects the percentage of skills mastered rather than an average of the competency of all the skills taught. Drawbacks of flexible assessment include limited time for in-class preparation, limited opportunities to review programming assignments and the increase in time required for grading.

KEYWORDS
mastery learning, specifications grading, flexible assessment, computer science education

https://doi.org/10.25891/en8j-cz52

A common challenge in creating a Computer Networks course is designing the course in a way that allows students with a variety of skill-levels and academic abilities to succeed. The tiered nature of the skills developed in the course means that students who stumble early are often unable to catch up. Lower performing students may master only a subset of the skills covered by an assignment, having only a limited proficiency of the remaining skills. Because each assignment builds on the full set of skills learned in previous assignments, a student who does not master an assignment starts the next assignment at a severe disadvantage. The cumulative nature of this disadvantage causes poor performing students to quickly fall so far behind that they cannot pass the class.

Borrowing techniques from competency-based learning (Gervais, 2016), specification grading (Berns, 2020; Nilson, 2014; Sanft et al., 2021) and mastery learning (Bloom, 1968; Garner et al., 2019; Keller, 1968), I have developed a Computer Networks course that allows lower performing students to pass. Programming assignments are often the component that prevents struggling students from passing the class. In response, I have created a flexible assessment pedagogy to allow the students to develop skills at their own pace. In this pedagogy, there are no due dates for assignments and I regrade assignments without penalty as often as the student likes. Students schedule an interactive, formative grading session with me whenever they feel their project is finished or has seen substantial improvement since the last grading session. This style of grading gives students immediate feedback and allows them to develop and build confidence.

Early results indicate that the flexible assessment pedagogy shows promise. It has increased the number of students who finish the course with a passing grade by 30 percentage points when compared to courses with a similar difficulty and audience.

There are some drawbacks and limitations to this technique. The flexible assessment pedagogy will not work for courses that are prerequisites for other courses because a student may not learn all of the prerequisite skills. Any skills they do learn, however, they master. The lack of due dates means that different students are often working on different programming assignments at the same time. This limits the amount of in-class review I am able to do for programming assignments. Perhaps the biggest drawback for the instructor is the amount of time spent grading.

Related Work

I have not developed radical new teaching methods. Rather, I have borrowed and adapted existing instructional approaches to fit the needs of my Computer Networks course and my students. The biggest influences for flexible assessment are competency-based education (Gervais, 2016), mastery learning (Bloom, 1968; Garner et al., 2019; Keller, 1968), and specifications grading (Berns, 2020; Nilson, 2014; Sanft et al., 2021).

Competency-based education (Gervais, 2016) focuses on students developing a predetermined set of competencies or real-world skills. The term is broadly used, but some hallmarks include unlimited regrading, self-pacing, student-directed learning, and frequent formative assessment. The largest difference between my Computer Networks course and a traditional competency-based course is that this course was not student-directed. I provided a schedule for course material, gave lectures, and assigned written homework that covered the theoretical aspects of the course content. Another major difference is that students were given 15 weeks to complete as many assignments as possible, unlike a traditional competency-based course where students have as much time as they like.

Mastery learning proponents posit that 90% of students can master the required learning outcomes of a course given sufficient time and help from the instructor. Common themes in mastery learning are that students work at their own pace, participate in formative evaluations with detailed feedback, and repeat assessments until they demonstrate mastery and, therefore, are ready to move on to the next topic. Like Bloom's (1968) Learning for Mastery and Keller's (1968) Personalized System of Instruction, my form of flexible assessment allows students to work at their own pace and provides formative evaluations with detailed feedback. In contrast to Bloom and Keller's models, my form of flexible assessment does not require students to achieve full mastery of one programming assignment before moving on to the next. Additionally, my course is time-limited, so I do not expect that 90% of the students will achieve mastery of all the required skills. Garner et al. (2019) provide a literature review of the use of mastery-based learning in computer sciences courses. The review includes only a small number of papers, which limits their conclusions. The authors conclude that most universities using mastery-based learning do so only for introductory computer science courses and that instructors often add traditional exams to provide summative assessment. The review finds that the primary motivation for transitioning to mastery learning is to improve learning outcomes in student populations with wide-ranging aptitudes.

Specifications grading (Nilson, 2014) creates a tiered assessment system. Each assignment grade is binary, pass or fail. Assignments are grouped into bundles or modules. A certain group of bundles must be passed before a student can earn a specific grade (e.g., bundles 1 – 2 for a D, bundles 1 – 4 for a C, etc.). This model includes a limited number of resubmissions for failed assignments. Modified specifications grading has been applied by other computer science instructors. Berns (2020) proposes a system called binary grading which is similar to specifications grading but allows unlimited resubmissions. Sanft et al. (2021) propose a modified specifications grading system with pass/fail assignments. Resubmissions are allowed, but incur a 10% penalty per retry. Sanft et al. show an improvement in student learning outcomes for middle to low performing students. The flexible assessment pedagogy that I developed allows both unlimited resubmissions and partial credit, rather than pass/fail, for programming assignments.

Concurrently with my own work, Lionelle et al. (2023) have developed their own flexible assessment model with a focus on large introductory computer science courses. This model includes formative assignments with no deadlines and no-penalty resubmissions. This model also adds a small number of summative assignments with a limited number of resubmissions and hard deadlines. Mastery of a topic is required for students to move to the next. This model has shown improved student performance in courses that follow the introductory courses, demonstrating that students are mastering foundational skills. In contrast to my own flexible assessment pedagogy, Lionelle et al. make extensive use of auto grading systems which limits the type feedback that can be provided to students. This limitation is a result of focusing on large classes where one-on-one formative assessments would be impossible.

Problem Addressed and Desired Aspects of a Solution

Students often fail Computer Networks unnecessarily. The skills and knowledge required for programming assignments in the course build on one another. That is, a student who does not successfully complete the first assignment cannot complete the second assignment since the skills developed in the first assignment are a prerequisite. Using a traditional approach to

teaching this course, a student who stumbles on the first assignment may never catch up and will ultimately fail the course.

With a perfect solution to this problem, a student who masters a subset of the skills taught in this class would pass. Students could learn a subset of skills at their own pace. And, any skill a student does learn, they learn well. Students who master all of the skills earn an A; students who master fewer skills earn a lower grade.

Course Description

Computer Networks is the study of the design and use of computer networks, with a focus on the modern Internet. The course focuses on the theoretical underpinnings of the modern Internet and the specific algorithms used to implement it. In particular, this course discusses client-server programming and its relation to the application, transport, network, data, and physical layer protocols of the Internet. Computer Networks is an upper-level elective for computer science majors and minors at the University of Lynchburg. Roughly half of our upper-level students opt to take this course. The content of this course is divided into theory and practice. The theory portion is focused on the foundational ideas and algorithms that define the modern Internet. The practical component, on the other hand, focuses on learning how to use the powerful network interfaces provided by programming languages, software libraries, and operating systems. Most graduates who work in computer networks will be developing software which uses the network, a practical application of this course. As good professionals, they should have a strong grasp of the theoretical foundations that power the networks they are using. Therefore, class time is divided between lectures, which cover the theoretical aspects, and hands-on-activities that cover the practical aspects. Students work in small groups to discuss and understand the theoretical concepts and they may work in pairs on the practical components.

Course Setup

Assignments for this course are divided into two categories: written assignments and programming assignments. Written assignments cover the theoretical aspects of computer network design and are intended to replace traditional exams. There are five of these assignments and they consist of four to five short answer questions. Students are given a week to complete each of these assignments. The typical student can complete them in a few hours. These assignments align with a traditional form of assessment: they have a fixed start and due date and students are only allowed to attempt them once, without regrades. These assignments compose 35% of the final grade.

The programming assignments provide students with practice writing software that use computer networks. There are three programming assignments and the typical student needs several weeks consisting of nine to ten hours per week of work to complete them. These assignments are the heart of the flexible assessment pedagogy. There are no fixed due dates except that they have to be completed by the end of the semester. Students are given freedom to decide how they complete these assignments. I also provide flexibility in the grading of these assignments; students can have these assignments graded as often as they like without penalty. These assignments comprise 65% of the final grade.

Through these programming assignments students build a simple client-server network file system, like a greatly simplified File Transfer Protocol (FTP) (Postel & Reynolds, 1985).

Each assignment builds upon the skills (and sometimes the code) students developed in the previous assignment. The first assignment focuses on building a network-based key-value store (see Appendix A for the full assignment). A key-value store is a simple database that stores key-value pairs. A user can ask a key-value store to save a key like "email-address" associated with a value like "somebody@example.edu". Later the user can ask the key-value store to retrieve the value for the key (email-address) to get back the associated value (somebody@example.edu). For students, the difficult part of this project is developing the network protocol to allow the data to be stored on one computer and accessed or updated from another computer. In the second assignment, students convert their key-value store into a network flat file system which is a file system that does not have folders or directories. At the end of the second assignment students have a network-based file system that allowed files to be added, retrieved, and appended to. The third and final assignment improves the reliability of the flat file system by adding features that could gracefully handle crashed or frozen clients and servers, protocol errors, or corrupted files.

I created these assignments thematically to allow students to demonstrate a mastery of a collection of skills that cover aspects of network programming. Program 1 contains a set of skills a programmer needs to build a rudimentary, text-based, client-server network application. Program 2 introduces the skills required to build a more complex client-server network application that can handle non-text data like video files. And Program 3 includes techniques to make a network application resilient to failures. As an example, see Table 1 for the set of skills developed by a programmer who completes Program 1.

Table 1
Skills Demonstrated by Completing Program 1

Program 1 Skills
Initiate and accept TCP connections
Send and respond to text-based commands
Develop a protocol to differentiate between commands and user data
Develop a protocol to differentiate between text-based errors and requested data
Develop a protocol that allows all text-based data to be transmitted (incl. special characters)
Gracefully recover from non-fatal errors in the server

Several aspects of these programming assignments are more flexible than their counterparts in a traditional networks course. Students are allowed to develop their own network protocol to govern communication between the client software and the file server. A network protocol defines the ordering of network messages (e.g., who initiates the communication) and the data format of the messages. In a more traditional networks course, the professor would define this protocol for the students. Students write the code for these assignments using the Java programming language. Java provides a myriad of ways to interface with the computer network, from low-level interactions using bytes to sending high-level programmer defined objects. I give the students the freedom to use any of Java's network interfacing tools. Again, in a traditional class, students would be given specific directions about how to use Java in their assignments.

The most flexible aspect of this approach is the lack of due dates or a rigid sequencing of the assignments. When a student feels that their project is ready, I grade it. Students earn points for each feature of the assignment that works to specification. If during the grading session a

feature does not work, the student is encouraged to spend more time working on it. When the student fixes the broken feature, I regrade the assignment. The student receives full points for each additional feature that works. This iterative approach to grading gives the student multiple opportunities to learn a skill with helpful feedback along the way. When a student feels that they have learned enough from an assignment, they move on to the next assignment, even if they have not developed all of the features of the previous one. This is a significant contrast to mastery learning and specifications grading. If the next assignment proves to be too difficult, the student can return to a previous assignment to reinforce the skills that it covered and earn more points. This is in stark contrast to the traditional rigid sequencing of a computer science course where a student is given a fixed amount of time to complete an assignment and must move on to the next assignment whether they are ready or not. The flexible sequencing of assignments in this course matches the nature of network software development quite well. A student must understand the rudimentary aspects of network protocols and network software development before they can attempt to build more complex real-world network applications. In a traditional networks course, if a student struggles with the first programming assignment it is impossible for them to complete any of the following assignments. This ensures that they fail the class. Using the flexible approach, students accrue network development skills at their own pace.

A key part of this flexible approach is setting clear expectations and standards for the students. At the start of each assignment, students are given the grading rubric (see Appendix B for the student's version of the rubric for Program 1). The rubric format is a list of three types of features: prerequisite required features, features that earn points along with their point values, and features that are not required. Students need to have all of the prerequisites completed before I will grade the assignment. These prerequisites are generally straight-forward and ensure that students do not violate the spirit of the assignment (e.g., the project must use a computer network). For each of the features that earn points, the rubric specifies how many points each feature is worth and provides general examples of the specific behaviors that must be supported (e.g., server should reject keys that are already in the key-value store). I include a list of features that are not required to prevent students from wasting valuable time building things that are beyond the scope of the project (e.g., the client does not need to have a graphical user interface). These rubrics are designed to help students focus on the most important aspects of each assignment and prevent surprises during grading.

On a broader scale, I also have a grading rubric for the full set of programming assignments. Students who earn all of the points on the first assignment earn a passing grade for the programming assignment portion of the class. Students who also earn all of the points on the second assignment earn a B –, and students who earn all of the points for all three assignments earn an A. I created this overall rubric to motivate students to complete all three assignments and to prevent them from being surprised by their final grades.

During the Course

During the course, the two most important parts of the flexible assessment process are student preparation and grading.

Student Preparation

Preparation for the first programming assignment is vitally important because the lack of due dates means I cannot host a post-assignment review session after the due date to discuss the proper way to complete the assignment.

After the first five weeks, most of the material the students need to complete the first programming assignment has been covered through the course lectures. In addition to the lectures, I use a class period to host a code-along during which we built a simple echo server and a National Institutes of Standards and Technology (NIST) time server client (Lombardi, 2002). A code-along is an interactive, in-class activity where the students and instructor write software together. During the code-along, I explain the feature we are trying to build and the Java classes we will use to build it. I then give them some time to try to build a NIST time server client on their own. After they spend some time working on it, we come back together as a whole group and they explain to me how to write the code while I type. With a little guidance from me, we are able to get a specific feature of the program working. Then we move on to the next feature and repeat the process. In this way, each student has two working network programs and several weeks of lectures on which to base the development of their first programming assignment.

Flexible Grading

The grading process is fairly straightforward and starts with the student. When a student feels that their assignment is ready to be graded, they schedule an individual meeting with me. During the meeting the student demonstrates the working features of their assignment. For each assignment there are specific test cases I use; these test cases cover aspects of the assignment students may not have considered. I record which parts of their assignment work to specification and which do not. Initially this grading process was slow for me, but by the end of the semester I had it down to about 10 to 15 minutes per student per assignment.

When a student's program correctly implements all of the features listed in the rubric, they have demonstrated mastery of a collection of related network programming concepts. In this way, a student that earns a C has demonstrated mastery of a subset of the skills required for this course. This approach to assessment differs from a traditional course where a C may mean the same as above or it may mean that the student only partially understands all of the course material.

In a traditional course, grading is often a form of summative feedback and can be viewed as pure assessment. Through the flexible assessment approach, grading gives students formative feedback that helps them better understand the skills they are learning. Students can use this feedback to improve their project and earn more points. This incentivizes quick integration of feedback to improve their skills. Another important feature is the interactive nature of the feedback. If a feature does not work to specification, we discuss why. Students sometimes misunderstand a core concept from the lectures and I take the time to re-explain it to them. Or they simply misunderstand the feature I am asking them to build. Summative feedback penalizes these misunderstandings. The formative feedback that I provide through the flexible approach allows students to expose and repair gaps in their knowledge without fear of it effecting their grade.

Without due dates, one of my largest concerns is academic misconduct. I do not want one student to finish the assignment and immediately give the answers to everyone else. To prevent

this, each grading session starts out with a conversation. The student explains the network protocol they developed for the assignment and answers some questions about the project. To prevent the student from feeling tricked, I give them these questions in advance. If the student cannot explain their protocol or answer the questions satisfactorily, we stop the grading session, no points are awarded, and I give them advice on how to prepare for the next grading session. This approach prevents students from sharing too much information about their assignments with their classmates. Students can give each other limited help, but they know that the help is only the start. Each student needs to cultivate a deeper understanding of why the project is built a certain way or they will not be able to answer my questions. Since this deeper understanding is a goal for this course, in my opinion, how it is acquired is somewhat irrelevant.

Results

During my first semester of using flexible assessment, the vast majority of students acquired some computer networking skills during this course. As shown in Table 2, by the end of the semester, 80% of students were able to demonstrate a complete collection of skills required to implement a simple computer networking program. And over half were able to acquire at least one skill at an advanced level. While just one student was able to demonstrate competency of all of the skills at the advanced level.

Table 2
Percent of Students whose Programs Met the Required Specifications

Programming Assignment	Fully met the specifications	Met 80% of the specifications	At least one feature met the specifications
Program 1: Key-value store	80%	86%	93%
Program 2: Flat File System	60%	80%	80%
Program 3: Reliable File System	6%	20%	53%

It is difficult for me to evaluate the success of this course compared to a more traditional approach because this was the first time that I taught Computer Networks and my university only offers one section of the course each academic year. As a result, I do not have data from a traditional networks class for comparison. However, I also teach the Distributed Systems and Operating Systems courses which are of a similar difficulty and have a similar audience. So I compared student performance in Computer Networks to student performance in my previous two sections of Distributed Systems and my previous two sections of Operating Systems. Distributed Systems uses a similar interactive assessment style, but with rigid due dates and without unlimited regrades. Operating Systems uses a traditional time-restricted, summative assessment system where students submit their work and I grade it without them being present.

In order to compare these courses, I defined two metrics: *completion percent* and *passing percent*. I defined completion percent to be the percentage of students who finished the course with a grade better than an F divided by the number of students who started the course. The passing percent is the percentage of students who finished the course with a grade better than an F divided by the number of students who did not withdraw from the course. In should be noted

that at my university, students who are struggling are allowed to withdraw during the first two-thirds of the semester, so passing percent should be higher than the completion percent.

Table 3
Pass, Fail, and Withdraw Data for Three Similar Computer Science Courses

Course	Students	Pass	Fail	Withdraw	Completion %	Passing %
Distributed Systems*	29	19	4	6	65%	82%
Operating Systems*	30	14	1	15	47%	93%
Computer Networks	15	13	1	1	87%	93%

* Two sections combined.

Table 3 shows the student completion percent for Computer Networks with flexible assessment was much higher than the rates for the comparable courses, but the passing rates were similar. Together, the comparable courses had a completion rate of 56% and a passing rate of 87%. Flexible assessment resulted in an increase of over 30 percentage points in the completion rate compared to the other two courses combined. This resulted in roughly five more students per section completing this course with a passing grade. The sample sizes here are, of course, small and it can be difficult to infer from these results whether future sections of the course will have similar results.

Table 4 shows the similarities and differences in grades across these three courses. Extracting meaningful patterns from this data is a bit more difficult. It appears that the flexible assessment approach is pushing students up from the bottom. It appears that withdrawals become Fs or Ds while Fs and Ds become Bs. This matches expectations from previous research which applied modified specification grading to computer science courses (Sanft et al., 2021). However, data from my course is likely too small to draw any conclusions about specific grades.

Table 4
Student Grade Distributions

Course	Students	A	B	C	D	F	W
Distributed Systems*	29	35%	10%	10%	10%	14%	21%
Operating Systems*	30	14%	27%	3%	3%	3%	50%
Computer Networks	15	32%	47%	0%	7%	7%	7%

* Two sections combined.

I also solicited student feedback on the flexible assessment aspect of Computer Networks. I asked students to evaluate different aspects of the flexible assessment features of the course. In particular, I asked if they found flexible assessment to be helpful to their learning process. They rated interactive grading, the lack of exams, and the lack of due dates on a Likert scale ranging from No Help to Great Help. The results appear in Table 5.

Students were somewhat split on the flexible assessment style of this course. Students liked not having exams and they had no complaints about the interactive grading. However, their feedback reflects that, overall, the class had a mixed view about the lack of due dates for the programming assignments. From their comments, it appears that some students felt that the absence of due dates caused them to procrastinate too much.

Table 5
Student Evaluation of How Much Each Aspect of the Course Helped Their Learning

Flexible Assessment Feature	No Help	A Little Help	Moderate Help	Much Help	Great Help
Interactive Grading	0%	0%	14%	29%	57%
No exams	0%	0%	0%	29%	71%
No due dates	14%	29%	14%	14%	29%

Percentages are based on the number of students that selected each category.

These results are similar to other studies where modified specification grading was used in computer science courses (Berns, 2020; Santf et al., 2021). To this point I would note that procrastinating students are the ones who are most likely to fall behind on the first assignment in a traditional course and are, therefore, the students who are most likely to withdraw from a traditional course. Through this alternative approach, while procrastination is still painful, it is not academically fatal.

Along with this criticism came some student suggestions for improvement. They suggested requiring the first programming assignment be completed before the course withdrawal deadline. They also recommended putting suggested due dates on the assignments as a guide.

Difficulties and Limitations

This instructional approach is not without its difficulties and limitations. As stated above, the biggest limitation is that flexible assessment will not work for a course that is a prerequisite for another course.

A difficulty I had not considered when I decided to use flexible assessment for this course is the limits placed on in-class preparation for assignments and post-assignment review. The crux of this problem is that students are really spread out in their progress through the programming assignments: I had students who completed all of the assignments with weeks to spare and others who were working on the first assignment right up until the last day. In my other classes, I often provide code-along days to prepare for difficult assignments. I could only do that for the first programming assignment in this course. Doing a code-along for the second or third assignments would have given away too much information to the students still working on the earlier assignments. Similarly, in my other classes I often perform post-assignment reviews after an assignment has been submitted and graded. In these reviews, I discuss areas in which the class as a whole struggled and better approaches they could have taken. There was never a point during the semester where all my students had completed the first programming assignment, so I could not hold a review session without giving away too much information to students still working on the assignment.

An unsurprising difficulty with teaching a course using an interactive grading system with an unlimited number of regrades is that the grading takes a long time. Early in the semester, grading sessions were taking 30 minutes per student. The typical student scheduled three grading sessions for each assignment they completed. Later in the semester, I made modifications which reduced the time to 15 minutes; this still resulted in spending 45 minutes with each student per assignment over the course of the semester.

Another difficulty I had not considered was justifying to the students the use of the grading system. I naively assumed students would love it and so there would not be any concerns. Nearly all of the students accepted that the system was fair, even if they thought it had shortcomings. However, I did have one student who procrastinated to the point of failing the course. He was, of course, unhappy and so was his father. In a traditional course, I could point to a series of assignments with Fs that led to the final grade. With a different grading system, it can be hard to explain to a concerned person that the problem is not "this one assignment" but the fact that the student waited way too long to try to complete one assignment and, therefore, failed. Not to mention the student did not even attempt the other two assignments. So, this approach requires some thought on the professor's part about how they will explain the grading process to students, parents, and administrators. It also requires support from the institution. If the administration does not encourage innovation or will not support faculty during a grade appeal, it may be better to stick to a more traditional assessment system.

Lessons Learned

As discussed earlier, grading takes a significant amount of time using the flexible assessment approach. During the course, I made some adjustments to speed up the grading process. First, I began to take and keep notes on each student's project including how they defined their protocol and their work on the program features. This reduced the amount of time spent at the beginning of the grading session reviewing how the student's project worked. I could simply ask what had changed since our last session. Second, I required a portion of the students' assignments to meet a specific Application Programming Interface (API), a set of function or method signatures. For example, the client-side of the key value store needed to have a method called *get* that took a key as a parameter, sent the request to the server, and returned the server's response. Prior to establishing a simple API, students had a myriad of ways of asking the client to retrieve a key's value, several of which made grading slow. These required APIs sped up grading considerably and provided some additional structure for the students.

Reducing student procrastination is a much more difficult task. I have several improvements I intend to make to address this problem in future courses. Based on student suggestions, I will add recommended due dates to each of the programming assignments. Moreover, I intend to incentivize students to conduct their first grading session before the recommended completion date and the rubric for an assignment will award a few points for doing so. It would be tempting to view these points as a late penalty. I intend to discourage that in two ways. First the points will not be worth a full letter grade; they will be worth a half-letter grade, separating a B from B– for example. Secondly, the assignment does not need to work to specification by the suggested completion date to earn these points. The student only needs to sign up for and attend the first grading session to earn these points.

My final improvement for reducing student procrastination consists of modifications to the student suggestion that I require the first assignment to be completed by the withdrawal deadline. I want students to be able to pass this class, albeit with a D, even if it takes the entire semester for them to complete the first programming assignment. So, instead of requiring students to complete the first programming assignment by the withdrawal deadline, I will require students to attend at least one grading session and strongly encourage them to complete the first programming assignment before mid-semester. If the student's first programming assignment is not graded prior to mid-semester, they will be withdrawn from the course for failing to make

satisfactory academic progress. The program does not need to work completely during the grading session and the student can schedule additional grading sessions for the assignment after mid-semester. However, if I grade their first programming assignment and it does not work to specification by mid-semester, the student will receive an F for their mid-semester grade, and I will email the student, their academic advisor, and the university's advising department stating that the student is on track to fail this course. These tiered requirements should create a strong incentive for students to attempt to finish the first programming assignment prior to mid-semester. Getting students to start an assignment is often the hardest part, and I am hoping that once they start the assignment they will follow through and finish it. Additionally, if the student still has not completed the first programming assignment by the withdrawal deadline (i.e., about three weeks after mid-semester), I will recommend to the student and their academic advisor that the student withdraw from the class. This policy should also help mitigate the problem with students who claim to be surprised that they failed at the last moment because they did not finish "one assignment." There will be a paper trail indicating that the student has known for weeks that they are in danger of failing.

Conclusion

Flexible assessment directly addresses a common difficulty encountered when teaching Computer Networks with overall positive results and limited drawbacks. I have adapted modern pedagogical techniques to improve the course which allows students with varied skills and abilities to pass. The heart of this approach is the flexible assessment of programming assignments which eliminates due dates and allows students to have their work graded and regraded without penalty. Flexible assessment also defines an interactive approach to grading which gives students immediate formative feedback and does not penalize initial failure. Using these techniques, I have increased the course completion percentage by 30 points when compared to similar courses. This approach also improved the learning outcomes for middle to low performing students. These results are similar to those found by researchers who modified specification grading for use in their computer science courses (Sanft et al., 2021). Overall, students liked the formative assessment style of this course, but some found the lack of deadlines to be an obstacle. Again, this matches other researcher's findings for mastery learning and specifications grading in computer science courses (Berns, 2020; Morais et al., 2014; Sanft et al., 2021). This type of flexible assessment works best for upper-level courses that are not prerequisites for other courses because a student can pass without mastering all of the skills; their grade reflects the number of skills mastered rather than an average of the competency over all the skills taught. Other drawbacks of flexible assessment include, limited in-class preparation and review of programming assignments and an increase in the time required for grading. Providing the students with a required API for each programming assignment will help improve the grading times. Students have noted that they prefer specific guidance about when projects should be completed to prevent procrastination.

Acknowledgments

I would like to thank Kevin Peterson of the University of Lynchburg and the JMSCE reviewers for their valuable feedback on early drafts of this article. I would also like to thank my students and the faculty in the Computer Science Department at the University of Lynchburg.

References

Berns, A., (2020). Scored out of 10: Experiences with binary grading across the curriculum. In J. Zhang et al. (Eds.), *SIGCSE '20: Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (pp. 1152 – 1157). Association for Computing Machinery. https://doi.org/10.1145/3328778.3366956

Bloom, B. S. (1968). Learning for mastery. *Evaluation Comment*, *1*(2), 1 – 11.

Garner, J., Denny, P., & Luxton-Reilly, A. (2019). Mastery learning in computer science education. In Simon & A. Luxton-Reilly (Eds.), *ACE '19: Proceedings of the Twenty-First Australasian Computing Education Conference* (pp. 37 – 46). Association for Computing Machinery. https://doi.org/10.1145/3286960.3286965

Gervais, J. (2016). The operational definition of competency-based education. *The Journal of Competency-based Education, 1*(2), 98 – 106. https://doi.org/10.1002/cbe2.1011

Keller, F. S. (1968) "Good-bye, Teacher…" *Journal of Applied Behavior Analysis, 1*(1), 79 – 89. https://doi-org.proxy.library.vcu.edu/10.1901/jaba.1968.1-79

Lionelle, A., Ghosh, S., Moraes, M., Winick, T., & Nielsen, L. (2023). A flexible formative/summative grading system for large courses. In M. Doyle & B. Stephenson (Eds.), *SIGCSE 2023: Proceedings of the 54th ACM Technical Symposium on Computer Science Education* (pp. 624 – 630). ACM Special Interest Group on Computer Science Education. https://doi.org/10.1145/3545945.3569810

Lombardi, M. (2002). NIST time and frequency services. National Institute of Standards and Technology. https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=105432

Morais, L., Figueiredo, J., & Guerrero, D. D. S. (2014). Students' satisfaction with mastery learning in an introductory programming course. *XXV Simpósio Brasileiro de Informática na Educação.*

Nilson, L. (2014). *Specifications grading: Restoring rigor, motivating students, and saving faculty time* (1st edition). Routledge. https://doi.org/10.4324/9781003447061

Postel, J. & Reynolds, J. (1985). File transfer protocol (FTP). Internet Engineering Task Force. https://www.ietf.org/rfc/rfc959.txt

Sanft, K. R., Drawert, B. & Whitely, A. (2021). Modified specifications grading in computer science: Preliminary assessment and experience across five undergraduate courses. *Journal of Computing Sciences in Colleges, 36*(5), 34 – 46. https://ccsc.org/publications/journals/SE2020.pdf

Appendix A

Programming Assignment: Simple Key-Value Datastore Server Feature Set (11 pts)

Description
A key-value store is a simple database. It can store key-value pairs. A user can ask a key-value store to save a key like "ITR" associated with a value like "itrhelp@lynchburg.edu". Later the user can ask the key-value store to retrieve the value for the key (ITR) to get back the associated value (itrhelp@lynchburg.edu). If you are familiar with maps or dictionaries, you can think of a key-value store as a program that behaves like a map or dictionary. You will be building a simple key-value store with a network API.

Grading
This project will be graded interactively in class or during office hours. When you are ready for me to grade it, let me know during class or sign up for office hours.

Required Features
- Server must accept serial connections, without restarting
- Server on cake/pie, client on desktop/laptop
- A client that demonstrates the features
- Client must have the following methods:
    - get(String key)
    - set(String key, String value)
    - put(String key, String value)
- You must be able to answer the following questions:
    - What is your message format?
    - What is your protocol?
    - How do you handle message framing?
        - How can you tell when you've received the entire message?

Features for Points
- 3 pts: *set* command
    - set a key-value pair
        - a get should be able to retrieve the value later
    - **server** should reject "" and null keys
        - server should tell the client the error
        - client should print the error message
    - **server** should reject keys that are already in the store
        - server should tell the client the error
        - client should print the error message
        - different error message from "" and null keys

- 2 pts: *get* command
    - given a key, provide the value
    - **server** should return an error if key is not in kv-store
        - server should tell the client the error

        - client should print the error message

- 2 pts: *put* command
    - provide a new value for a key already in the kv-store
        - replaces the old value
        - a get should be able to retrieve the new value later
    - **server** should return an error if key is not in kv-store
        - server should tell the client the error
        - client should print the error message

- 3 pts: All UTF-8 characters are allowed in keys and values
    - including control characters like \n and \r

- 1 pt: Submitted code to Moodle as a zip file

Not Required Features
- Persistent connections
- Key-value store data does not need to be persistent
    - if the server is terminated, the data is lost
- Concurrent connections
- Keys or values that are not UTF-8 strings
- A nice user interface for the client
    - your client can just be code with no user input
    - it should output enough information so that I can be sure your server works

Appendix B

Rubric Example: Simple Key-Value Datastore Server Rubric Feature Set (11 pts)

Prep notes
- None

Required Features (Double check these before starting)
- Server must accept serial connections, without restarting
- Server on cake/pie, client on desktop/laptop
- A client to that demonstrates the features

Questions to ask before starting
- What is your message format?
- What is your protocol?
- How do you handle message framing?
    - How can you tell when you've received the entire message?
    - How can you tell when one message ends and another begins?

Features for Points

- 2 pts: simple set-get
  - set “dog”->small
  - set “mouse”->alive
  - get “dog”
  - get “mouse”

- 2 pts: invalid sets
  - set “”->small & set null->small
  - set “dog”->large
    - different message than “” and null keys
  - *ask to see code where **server** handles invalid keys*
    - *no points if server is not the one that handles invalid keys*

- 1 pt: invalid get: key not in the kv
  - get “cat”
  - *how can client differentiate between error messages and non-error messages*
    - *what if my value is an error message?*
  - *ask to see code where **server** handles invalid keys*
    - *no points if server is not the one that handles invalid keys*

- 1 pt: put and get
  - put “mouse”->”dead”
  - get “mouse”

- 1 pt: invalid put: key not in kv
  - put “cat”->”small”
  - *ask to see code where **server** handles invalid keys*
    - *no points if server is not the one that handles invalid keys*

- 3 pts: All UTF-8 characters are allowed in keys and values
  - set-get: "split\r\nkey" -> "split\r\nvalue"
  - set-get: "double\r\n\r\nsplit" -> "double\r\n\r\nsplit"
  - set-get: "period\r\n.\r\nsplit" -> "period\r\n.\r\nsplit"
  - *ask how messages encoded*
    - *if using Strings, ask them how they delimited between cmd, key, value*
    - *what happens if that delimiter appears in key or value*
    - *if it cannot appear in key or value, no points*

- 1 pt: Code in Moodle