How We Have Applied Grading For Equity: An Experience Report

Andrew Berns, J. Philip East & J. Ben Schafer
Computer Science Department
University of Northern Iowa
andrew.berns@uni.edu   philip.east@uni.edu   ben.schafer@uni.edu

Introduction

As with most faculty we seek to improve our students' learning. Over the years we have tried various tools and techniques. Grading has long been problematic (particularly for Philip) and often been the subject of exploration for possible improvement (East, 1998, East, 2001, East & Schafer, 2005). Not surprisingly we read *Grading For Equity* (Feldman, 2019) with much interest. Philip immediately tried to adopt the suggested practices and Andy and Ben shortly followed suit.

Grading For Equity (GFE) starts with the assumption that grades should reflect student capability at the end of the course. So, grades should accurately reflect student capability (and only student capability). Additionally, grades should be bias-resistant. For example, a student who already possesses substantial understanding and one who started with no background should receive the same grade if they have the same capability at the end of the course. Traditional grading practices tend to allow biases in various ways. Finally, grades should motivate students rather than demotivate and discourage them. Traditional grading practices are meant and thought to be motivating but are not. Feldman (2019) presents a very good research- and logic-based argument that traditional grading practices are not accurate, are biased, and are not motivating; he suggests alternatives to them that make grading more accurate, bias-resistant, and motivating.

The rest of this paper will provide an overview of *Grading For Equity*. We will discuss: 1) ways in which Andy has attempted to apply GFE in his current upper division courses, 2) Ben's approaches to applying GFE in his teacher preparation courses, 3) some pre-GFE practices that allowed Philip to readily jump on the GFE bandwagon, 4) our personal reactions to GFE, and 5) suggestions for implementing GFE in your own instruction.

Pillars of Grading For Equity

Feldman (2019) describes his vision of equitable grading as having three pillars: equitable grades are accurate, bias-resistant, and motivational. The discussion below indicates issues with traditional grading practices and briefly identifies some alternatives.

Accurate

Again, accuracy of grades refers to the assessment of student capability at the end of the course with respect to explicit course outcomes. Grades should accurately reflect such capability. Inaccuracy in traditional grading practices arise from:

- Unequal grade ranges
  Traditional A-D grade ranges are 10 points each while the F range is 60 points. Our intent may be that students need to "know" at least 60% of our content but all 60% scores probably means the student doesn't really know much of anything.
- Broad grade ranges
  Possible scores on individually graded items often are 10, 15, 20, 25 or even 100 points. We often find it difficult to consistently distinguish whether a particular submission should result in a score of 6 or 7 or even 8. Consistently awarding the same score for similar performance is also difficult. A range greater than 10 would make this even more difficult. Such possible inconsistency implies inaccuracy.
- Use of the zero
  A zero can indicate a student has demonstrated little or no capability on some particular assessment. But it is also often used to indicate the absence of an assessment or that a student was deemed to have cheated on the submission. Any use other than the little or no capability assessment makes grading inaccurate.
- Weighted averages
  Having categories of graded activity and using weighted averages to get an overall score causes results that are likely inconsistent with our intentions. It is possible, for example, that one student did well on homework and another student did well on exams. Would equal overall scores truly suggest equal capability? Similarly, different instructors (or the same instructor in different terms) might weight categories differently. Should two students with the same exact scores get different grades due to category weighting?
- Grading behavior
  Grading behavior occurs when we include attendance, participation, early submission bonuses, late submission penalties, etc. in our grading. Such actions result in inaccurate grades as the grades no longer represent student capability on course outcomes.
- Grading group work
  The grade on a submission from a group does not accurately reflect an individual's capability. Group work can be a great learning activity but we need to identify alternatives for assessing individual performance.
- One and done assessments
  In traditional grading practice, when a student fails or performs poorly on an initial assessment they are mostly up the creek. Students who later develop that capability retain the early poor score and their overall grade does not accurately reflect their capability.

To grade accurately requires that we only count individual performance on assessments specifically designed to demonstrate capability with desired course outcomes. Additionally, students who may not succeed on the first assessment must be allowed chance(s) to repeat the assessment activity or later alternative assessments.

Bias Resistant

Almost no one intends to teach inequitably. It turns out, however, that inequity is embedded in traditional grading practice. Nonacademic performance gets included in grades through behavior, academic background, and environmental/life circumstances, as discussed below.

- Behavior
  Good intentions usually influence inclusion of behavior in our grading—either poorer students will get better grades or students will be encouraged to perform in a way that should enhance their learning. Such instances include: attendance and/or participation, late penalties, early submission bonuses, cheating penalties, etc.
- Background
  Background—both content-specific and general—often affects student grades. Students come to us with varying degrees of: exposure to our content, study skills, ease/speed of learning, reading skill, etc. Homework scores, for example, will be biased in favor of "better" students and those with prior knowledge. Allowing extra credit will have a similar bias. Having only a single opportunity will also favor these same students. Cultural background can also impact student grades under traditional grading practice. For example, speaking up in class might be considered "acting white" in Black culture or showing disrespect (disagreeing with the instructor) in Asian culture. Some cultures may encourage helping peers by sharing. Establishing rules regarding such things that have an impact on grades can force students to choose between their culture and their grade.
- Environment
  Students' personal environment or life circumstances can also affect student grades under traditional grading practice. Personal environment includes such things as physical health, mental health, family situation (hospital visits, funerals, childcare, etc.), the need to work, extra curricular activities, etc. These come into play when we include attendance and participation, use late penalties or early bonuses, etc.

Strengthening grading against bias is relatively straightforward. Don't grade homework (but remember that feedback is still important). Perhaps more succinctly, only include assessments of outcomes when grading. Additionally, provide multiple opportunities to demonstrate learning regardless of when (or how) the learning occurred.

Motivational

When we practiced traditional grading techniques, we thought we were motivating students to learn. It turns out, however, that contingent extrinsic rewards (do this to get that) "may reap short-term results but can smother intrinsic motivation and its benefits and cause undesirable side effects" (Feldman, 2019, p.155). Additionally, "when the task requires creativity and expansive complex thinking …, extrinsic motivation *lowers* performance" (Feldman, 2019, p.155). GFE practices center student focus on learning and student capability due, we think, to the elimination or reduction of "points". Finally, the use of multiple tries on assessments demonstrates and supports the strengthening of a "growth mindset" (Dweck, 2006).

Changes to Grading Practice

Feldman (2019) provides a table (p. 72) that indicates the traditional grading practices to avoid and alternative grading practices to adopt to support the pillars of accuracy, bias-resistance, and motivation. It is reproduced with some annotations below.

Accurate Our grading must use calculations that are mathematically sound, easy to understand, and correctly describes a student's level of academic performance.	Avoiding zeros—they should indicate performance, not lack of assessment or cheating penalty
	Minimum grading—actually refers to making grade ranges equal, i.e., minimum grade would be 50 in a 90/80/70/60 grading scale
	0-4 scale—automatically includes above two items. Lesser scales are reasonable
	Weighting more recent performance—allow retakes and redos
	Grades based on an individual's achievement, not the group's
Bias-Resistant Grades should be based on valid evidence of a student's content knowledge, and not based on evidence that is likely to be corrupted by a teacher's implicit bias or reflect a student's environment.	Grades based on required content, not extra credit
	Grades based on student work, not the timing of the work
	Alternative (non-grade) consequences for cheating
	Excluding participation and effort
	Grades based entirely on summative assessments, not formative assessments (such as homework)
Motivational The way we grade should motivate students to achieve academic success, support a growth mindset, and give students opportunities for redemption.	Minimum grading and 0-4 scale
	Renaming grades—e.g., poor, marginal, okay, good, excellent
	Retakes and redos
The ways we grade should be so transparent and understandable that every student can know her grade at any time and know how to get the grade she wants.	Rubrics
	Grades bases on standards scales, not points
Equitable grading distinguishes and connects the means for learning effectively the "soft skills," the practice, the mistakes, from its ends—academic success, and utilizes the broad universe of feedback and consequences, of which only one part is a grade.	Standards based gradebooks
	Emphasizing self-regulation
	Creating a community of feedback
	Student trackers

We think the GFE practices can be summarized in three practices with respect to what to items to grade and how to grade them.

- Grade only individual summative assessments of course outcomes
  Don't grade homework. Don't grade behavior. Avoid punishments. Find individual alternatives to group work assessments.
- Use limited grading scales
  The 0-4 scale is familiar and associates well with grades. Other shorter scales can be useful. Use names for performance rather than numeric scores, e.g., poor/marginal/okay/good/excellent, marginal/okay/good, not-competent/competent or use rubrics that provide more detail about the depth of student understanding or how well they can apply it.
- Provide multiple opportunities to demonstrate capability
  Plan for retakes or redos. If later assessments address earlier outcomes, allow that performance to replace earlier performance.

We recently realized something we had not gleaned from Feldman (2019) or noted elsewhere—grades are categorical. There is no 100%. For rote knowledge, a student either knows it or does not. For understanding, a student may express great understanding but there is almost certainly something that wasn't mentioned, so we may say the student's response was excellent but it almost certainly was not 100%. Traditional grading uses points to place performance in the appropriate score range/category.

Statistics tells us that summing, averaging, etc. categorical data is inappropriate. So, when we have graded all the items submitted by our students, we need to do something other than summing and averaging the individual grades. If we wish a 60% threshold, perhaps 60% of the assessments need to have been at the highest level or highest two levels or … Or we might start by considering what A level work is. If individual assessments were all A-F, we might expect all As, or 80% As with the rest Bs, or … In any case, identifying courses grades will not be as simple as summing the weighted averages we've generally used in the past.

Course Planning and the Side Effects of Grading For Equity

GFE requires the initial determination of course outcomes. Doing so is familiar to CS faculty who are experienced in top-down design. Educationally, this is referred to as backward design by Wiggins and McTighe (2005). CS folks readily use top-down design when programming but don't (at least "we" did not) typically use it for instruction. We were used to thinking in terms of content, not outcomes, and to building course plans from the bottom up according to prerequisite structure (or as the text organized it). So, performing the task of identifying course outcomes is new and may be a bit difficult the first time or two. However, it does come with some nice benefits.

The instructional design process begins with formulating the one or very few general goals for the course. The goal(s) are in terms of capabilities, what students should be able to do, not content or knowledge. Next the general goals are broken down into constituent capabilities. The refinement continues generating sets of outcomes until assessments for each becomes reasonable. For example, one set of outcomes under the general goal of "develop a correct program for a simple problem requiring less than a page of code, relating to selection (if) might be 1) evaluate/trace if statements, 2) modify (to use or avoid `and` and `or` operators or to use or avoid nested `ifs`, and 3) provide code for given selection tasks. Once the outcomes design is completed, assessments and instructional plans can be completed.

The side-effect benefits of outcomes explication now become apparent. The list of outcomes provides an outline for the instruction to teach them. It also serves as a study guide to share with the students. The outcomes list/outline provides a template for generating the multiple assessments needed for retakes and redos. Our experience is that the process of explicating outcomes makes us more aware of the content of our courses. That helps us consider alternative instructional strategies as we plan our instruction. The outcomes themselves help focus on the instruction, on instilling/developing the desired capabilities.

A final benefit of backward planning to develop outcomes and assessments is that it provides a mechanism for course evaluation. The record of student performance of each of the outcomes provides direct evidence of success in the course. Poor performance on any of the outcomes can indicate where efforts for improvement need to be focused. Acceptable or good performance across the board provides evidence of success.

Andy's GFE Experience

I have always disliked grading and have spent significant time reading and thinking about practices to make grading more bearable, including extensive use of autograding software and approaches such as specifications grading (Berns, 2020). When Philip first mentioned Grading For Equity, I was intrigued. As I learned more about it, I saw how its application would help make grading easier *and* provide my students with a more equitable educational experience. Since being introduced to GFE, I have worked at modifying several classes to adopt its principles. In this section, I discuss my experiences in several of these courses: software engineering, software verification and validation, and theory of computation.

Software Engineering

For several semesters now, I have been working at integrating grading for equity in an introductory software engineering course I teach at the University of Northern Iowa. Course enrollment is usually around 35 juniors and seniors in one of our undergraduate computer science degree programs.

When I begin designing a course, I like to start with a "big question": a singular question that I want students to be able to "answer", and which guides all of our work and activities. For software engineering, my "big question" is:

What can we do to ***quickly*** *and* ***repeatedly*** *create* ***high-quality*** *software at a low* ***cost****?*

From this big question, I can derive general course outcomes. These are constantly changing as I work to improve my teaching and my assessment, but for this semester my general course outcomes are that students should be able to:

- explain how the practices of several proven software life cycle models help manage the complexity that arises with large software projects.
- apply a proven software life cycle model to systematically create and maintain a software product.

In order for grading to be accurate, motivational, and bias-resistant, I need to make sure that students are being evaluated on their abilities with respect to one of the outcomes, and in order to create meaningful classroom experiences, I need to ensure the things we do in class help students reach demonstrable competence in the identified outcomes.

To see the process of moving from a high-level outcome to tangible assessment and instruction, consider the first outcome of the software engineering course: students should be able to "explain how the practices of several proven software life cycle models help manage the complexity that arises with large software projects". As a first step, I divided this broad outcome into several modules: plan-driven software processes, and change-driven software processes. For each of these modules, I identified assessments that would let me know students have become competent in the software process being discussed. For instance, I have a quiz question that gives students a sample scenario and then asks them to select the appropriate software life-cycle model. These questions are open-ended and could go either way—plan-driven (waterfall) or change-driven (extreme programming (XP)). What is important is that the student's response contains an accurate and relevant discussion of the pros and cons of each model, along with a conclusion that logically follows from the points they identified. I use binary grading for this problem: either the student demonstrated enough to show me competence, or they did not. If the student is not competent yet, I simply discuss the results with the student, and then administer a new quiz with a different scenario. Many students succeed on their first attempt, and almost all succeed after two attempts.

Finally, I ask myself what I can do in class to better prepare students to complete the assessments. For the process selection question, for instance, we discuss in class the pros and cons of each approach, discuss as a group the appropriate selection of life cycle for several scenarios, and then have students make their own scenarios and justification for the life cycle model they choose. This activity usually generates a lot of good in-class discussion and I think it helps students make these abstract process models more concrete.

This general process is repeated for the other course outcomes. Course preparation becomes the process of transforming outcomes into assessments, and assessments into learning experiences. I liken it to the need for requirements traceability in a software project: for every outcome, I can identify the activities and assessments we use, and for every activity or assessment, I can identify the outcome it is for.

The course consists of several types of these "learning activities". Activities can consist of quizzes (as mentioned above) or writing assignments where students read a selected article and then write a few pages "demonstrating they've thought about the contents of the article". I include a few sample writing prompts but encourage students to write about something else if they wish. Every quiz or writing assignment that is not successfully completed can be re-attempted, either taking new quiz questions or revising their work for the writing assignment. At the end of the semester, the total number of activities successfully completed is used to compute their final grade. For instance, if 15 activities have been assigned, 13 successfully completed activities is an A, while 10 is a B, and so on.

Software Verification and Validation

I have also been working to incorporate equitable grading practices into a software verification and validation course offered to upper-level computer science majors at our institution. During this course, we work to answer the "big question" of:

> *What can we do to deliver **better quality** software?*

To answer this question, the class focuses on two major outcomes. In particular, students should be able to:

- Define various types of defects and discuss how and when they enter different artifacts of a software project, including during specification, design, and implementation
- Apply proven practices for verification and validation to efficiently discover defects in various software artifacts

From these objectives, we divide the course into five units, one discussing software defects and their classification, and then four examining how we can ensure a quality software product is delivered either through detecting faults (white-box and black-box testing), through static program analysis, and through formal specification languages. For each of these units, I have three types of assessments: worksheets where students are asked short answer questions about materials covered in lecture, reading assignments where students are asked to read and reflect on some new (to them) topic in software verification and validation, and programming assignments where students are asked to implement a technique that we discussed in class. This mixture is to help meet the outcomes: students must be able to define elements of verification and validation *and* apply modern techniques.

For an example of the assessments, consider the static program analysis unit. The three assessments are (i) a short worksheet chiefly over the pros and cons of various static analysis techniques, (ii) a reading from the *Communications of the ACM* on formal methods at Google, and (iii) a programming assignment asking students to annotate a card deck implementation using either JML or ACSL. These assessments then guide the activities we do in class: we use traditional lecture to cover the big ideas behind static analysis; we use a few class periods to work with specification languages; we spend a little time discussing as a group the reading, both what it was about but also things to think about or note while reviewing it.

Grading is again done on a pass-fail basis. For reading assignments, I am merely ensuring that the paper demonstrates they read the paper and thought about the topic (and met the minimum word count, a requirement I found necessary after several short paper attempts). It is important to note that I am *not* looking for a formal well-formatted paper, I am only looking for their demonstration of critical thinking. For implementation assessments, I am looking not only for basic correctness of the program, but also the correct application of the technique (e.g., appropriate pre-conditions, post-conditions, and invariants for the card deck implementation). For the worksheet, I am of course looking for the correctness of their answers.

As with software engineering, the final grade in the class is determined by how many assessments are completed successfully. There are more assessments "assigned" than are required to pass the course, so students get some agency in selecting the items that interest them the most. Even at the A level, there is one assessment that a student may skip.

Theory of Computation

I have also worked to GFE into our undergraduate theory of computation course. For this course, I use the following "big question":

> *What are the **capabilities** and **limitations** of computers? That is, what **can** and **cannot** be computed?*

From this question comes our course outcomes. Students will be able to:

- Explain several different models of computation, identifying the commonalities and fundamental aspects of computation, and discussing their relevance to modern applied computation
- Define a computational problem and classify it with justification as uncomputable, intractable, or tractable

Tied into these high-level outcomes are 31 "individual objectives", which are basically quiz questions that I will ask students to demonstrate their abilities. These objectives are grouped by the typical categories for an undergraduate theory of computation course: regular languages (finite automata), context-free languages (pushdown automata), and computable languages (Turing machines). A full list of these objectives can be found at:

> http://www.cs.uni.edu/~adberns/courses/past/cs3810-s20/objectives/

Consider, for example, the unit on context-free languages. As part of this unit, I might ask students to determine if a given pushdown automaton accepts or rejects a particular string, or to prove that a given grammar is ambiguous. As with my other courses, there is an explicit correspondence between the tools used for assessment and the high-level outcomes for the course. Similarly, there is a correspondence between the learning activities (frequently in the traditional lecture format but more focused on the individual objectives, making the connection explicit to students before class) and assessments.

At the end of the semester, grades are determined again by how many objectives they have completed. While there are 31 individual objectives, only 27 are required for an A, allowing students to have a little flexibility in how they earn their desired score. A student might choose to skip questions on the pumping lemma for context-free languages, for instance, and can still earn an A in the course.

Ben's GFE Experience

My experience with Grading for Equity came at the same time that I was transitioning from teaching CS majors to working with in-service and pre-service teachers in UNI's teacher prep program. Because of this I was already facing decisions about how I would be designing assignments and assessments linked to very different course outcomes. Given that perspective, it was actually very easy for me to embrace GFE and see how I could incorporate it into my newly designed courses. While I had spent years embracing the "pile of points" style of grading, I did not have that long of a history with most of these particular courses.

I teach six different courses in our teacher prep program:

- Programming Environments for Elementary Education - a required course for elementary education majors earning the math endorsement. One section each semester averaging approximately 20-25 undergrads per section.
- Fundamentals of Programming - a required course for math education majors at the secondary level. Two sections each year. One undergraduate section averaging 25 students. One graduate sections (online) averaging 12 students.
- Teaching and Learning Programming –additional programming experiences framed within the study of programming pedagogy. For those who elect to add the CS endorsement to their license. Two sections each year. One undergraduate and one graduate (online), each averaging 12 students each.
- Foundational Concepts of Computer Science – a broad topics course similar to the non-programming topics that are part of the Computer Science Principles coursework. For those who elect to add the CS endorsement to their license. Two sections each year. One undergraduate and one graduate (online), each averaging 12 students each.
- Data Structures and Algorithms – an "advanced" programming course blending together various elements of the two traditional CS majors courses. For those who elect to add the CS endorsement to their license. Two sections each year. One undergraduate and one graduate (online), each averaging 12 students each.
- Methods of Computer Science – the capstone pedagogy course. For those who elect to add the CS endorsement to their license. Two sections each year. One undergraduate and one graduate (online), each averaging 12 students each.

While each of these classes has its own set of grading structure under the GFE "umbrella" I can explain much of my implementation philosophy by focusing on the two types of courses I teach.

That is, programming courses required for a particular education audience, and elective, "endorsement" courses.

The introductory programming courses are required for students who want a mathematics endorsement at either the elementary or secondary level. While the course learning outcomes are very different, much of the base grading and the overall challenges (an audience with no programming experience and no choice in being in the class) are the same.

Programming Environments for Elementary Education (PEEE)

PEEE is a traditional, 15-week course taught every fall and spring semester. The main learning outcomes for the course focus on understanding the key concepts of CS, learning about the state standards for elementary CS, and in the process, learning how to teach CS integrated into existing elementary curriculum. The course includes studying the Code.org CS Fundamentals curriculum, general programming in Scratch, understanding CS and Computational Thinking (CT) and the state standards, and mechanisms for integration. A typical class week consists 1 or 2 days of instructor led lessons, a hands-on small group activity, and a wrap up individual reflection or programming assignment.

While activities completed in class, either as an individual or with a small group/partner are collected as formative assessment, they are not graded. The only activities formally graded in the course consist of seven programming assignments and one lesson plan activity – completed individually and outside of class – and three "competency demos" (similar to a traditional exam) given live and in class.

Everything in the class is graded on a scale from 1-4 where loosely speaking

- 1 means you turned something in but you really haven't addressed the assignment
- 2 means you submitted an assignment that showed significant learning but had some deficiencies that need to be addressed
- 3 means you showed competence. I feel you understand the material.
- 4 means you showed competence/understanding as well as demonstrating initiative and growth beyond the base requirements.

The programming assignments and the lesson plan activity are all assigned with a published "grading contract" that are somewhat similar to a rubric. Each assignment has a set of expectations that need to be met to show competence (a level 3 score) with the current topic and multiple "extensions" that can be added (a level 4 score). These are typically things that go beyond base competence and allow the students to show they were thinking about doing something more than "checking the box."

Each assignment has a base "deadline" (typically 2-3 class sessions after it was assigned) by which I ask that it be completed, but I do not penalize students for missing this deadline. Shortly after the deadline I assess all of the student deliverables and assign each student a grade – almost always a 2, 3, or 4. Students who receive a 2 are told why they did not yet meet "competence" and strongly encouraged to improve their deliverable. Students who receive a 3 have the option of improving their code as well or meeting with me to make a case for why they felt their deliverable deserved a 4. This process is ongoing over much of the course. I will typically accept revisions up until the last week of the semester.

Each competency demo is given in class using our campus LMS. They consist of a wide variety of questions ranging discussing both the base CS concepts (vocabulary/definitions, identifying elements of programming in examples, etc.) and connecting these to teaching and learning ("Explain a real-world example you would use with your students to explain loops." "A student in your class is trying to understand the difference between conditionals and events. How would you help them understand the difference?"). Each question is graded on an individual scale or 1-4. Then all of the questions are assessed collectively and **two** scores (again, 1-4) are given for each competency demo – one based on assessment of CS concept questions and the other for CS pedagogy issues. Students who want to improve their scores are asked to schedule a meeting with me (approximately 20 minutes) to talk about their responses on the first attempt and what I feel that they need to include in an answer to move up to a 3 or a 4. These conferences typically take place over the course of 7-10 days after the initial competency demo and a formal retake is scheduled for all who want to attempt to raise their grades.

At the end of the semester students have earned 13 scores from 1-4. I have played with several iterations of assigning final grades from this. Currently I am using the following:

- All grades are "competent" (3s or 4s) and average score of at least 3.5 is an A.
- No more than one grade of a 2 and an average score of at least 3.0 is a B.
- No more than two grades of a 2 and an average score of at least 3.0 is a C.
- All work submitted and an average of at least 2.5 is a D.
- Any work not submitted is an F

Some notes on this:

- The advantage of asking students to meet with me and talk about their responses on the Competency Demos is that I have an opportunity to see what they are thinking. I typically ask them to explain their answers to me. In about a third of the cases I discover that they really do understand the concept/question but just gave a misworded or misguided answer. In other words, my assessment of their understanding was incorrect. In these cases, I often change the grade and do not require students to retake the CD.
- I have never once had a student do worse on the retake. The worst-case scenario is that they do about the same and do not improve their overall assessment. But most do significantly better.
- I actually am fairly strict with my assessments. That is, I don't feel I have to give a student the "benefit of the doubt" if their response is on the fence. If I am doubtful if a student fully understands a question, I will give them a score of a 2. But we BOTH know that they have a chance to better understand what I want them to know, learn it, and then show me on a second attempt.
- Most students honestly feel empowered by this process and are less stressed out. They recognize that if they misunderstand something or fail to adequately prepare for the way I write questions, that they will have a chance to adjust and legitimately learn the material.
- This process seems time consuming. It really isn't. I spend less time grading assignments due the simplified nature of the grades assigned (1-4). Therefore, I can afford to give the

students who "struggled" a little bit of one-on-one time and the benefit of allowing them to retake the competency demo.

- The number of retakes required drops significantly as the course goes on. Students learn that I have high expectations for their responses or deliverables and work hard to hit it the first time. But they do so with the knowledge that if they fall short it won't ruin their chance to earn a good grade because they have an opportunity to "fix" the deficiencies.
- In the end, 85% of my class earns an A. 10% earn a B or C. 5% fail because they stop coming to class. No one has ever legitimately finished the course but earned a D or F.
- While that might seem like grade inflation, I honestly feel that most students have learned more than they did in my "pile of points" days. Students have the opportunity to re-visit the material and honestly relearn it.
- I rarely give pluses and minuses. That might be surprising when you consider that seven 4s and six 3s is an A while six 4s and seven 3s is a B. However, I almost never have students straddling that division. Most students have a division closer to 10 and 3 which is either clearly an A or clearly a B depending on whether that's ten 4s or ten 3s. While I am not formally running "contract grading" the expectations for the grades are very clear. Most students decide early on that they are going for an A and will rework any 3s up to a 4, or they decide that a B is just fine with them and they stop when they hit the 3s. But in all cases, they have made a very conscious decision about their grade and are meeting it.

Fundamentals of Programming (FOP)

FOP is a similar class but addressing secondary CS standards and, therefore, going into much more depth of content and expectations. This course has a similar overall structure to PEEE in that a typical week consists of 1-2 hours of "lectures" (a combination of pre-recorded video and in-class lessons), 1-2 days of small group, in-class practice problems, and a set of outside of class practice problems. All in all, the course provides significantly more opportunities to practice programming. To accomplish this, we use an Autolab server (autolabproject.com) to automatically test student code. We provide students with ~8 practice Python programs each week. The Autolab server runs automated tests on student code to allow them to receive feedback on the functionality of their code. Students are strongly encouraged to complete as many of these as they can each week to gain practice and skills. Most of these are not included in their course grade. They are there for practice only. Instead, approximately every two weeks we take one class session to have students complete one or two programs (depending on the size and nature of where we are in the course) as an in-class competency demo. If all programs pass the Autolab tests by the end of class, then the student receives a 4 for the competency demo. If one passes and one does not, I hand grade the one that didn't pass to see why. If the problem is very close with a minor issue then I talk with the student about what happened (frequently it's a small thing and it is the time limit that prevented them from figuring it out) and the grade is a 3. In other cases, the code is hand graded and assigned a score of 1 or 2. Overall course evaluation is very similar to that listed above with PEEE.

Comments on Other Courses

The four remaining courses listed are all courses that are required to allow students to earn the secondary computer science endorsement on their teaching license in Iowa. The participants in these classes have all elected to continue in the program to earn this endorsement and have, for the most part, already completed the introductory programming course which uses GFE. Between their prior experiences with GFE and the shift in course materials, these courses have a different look/feel to them.

First of all, each of these courses is heavily tied to weekly readings – either from a textbook or from provided outside sources - and/or thought prompts about the weekly topic. A typical "unit" is 2-3 weeks and requires multiple readings and 4-7 writing prompts due at the start of class and that guide the daily discussion(s). While these papers are formally collected, they are assigned a simple binary grade of 1 or 0. Students are told that a score of 1 means that their responses are competent (a 3 or 4 if the same question had been part of a Competency Demo) while a 0 means that I felt they were lacking in details or specifics to earn one of these "competent" grades. Having said that, these scores are recorded but NOT part of the overall course assessment. They are simply a way to give students course-grained feedback on their progress. Each of these is officially due at the start of class because they are used to drive class discussion, but I accept them at any time during the unit. Because grades are not recorded for overall course assessment purposes, I do not traditionally ask students to revise papers that receive a 0 (although I will assess them if they do).

At the end of each unit (again, every 2-3 weeks) students complete a competency demo. Each is 2-5 questions, many of which are directly or indirectly based on previous questions used on daily reflections. These are individually graded using a 1-4 scale and combined to a single 1-4 grade similar to what was done in the programming courses. Sometimes these are done "live" in class. Other times the CD is handed out on Friday and returned to me on Monday. That choice depends on the nature of the material and the coursework as well as whether I consider this to be "open book" or "closed book." As with the CDs in the programming courses, any CD less than a 4 can be retaken a second time with a strong encouragement that students meet with me to discuss why they didn't have a 4 the first time around to help focus their preparation for the retake.

Final course grades are assigned similarly to the grades assigned in the programming courses.

In conclusion, I have been very impressed with how students respond to this overall process. In short, I notice that:

- They feel much more in control of their own learning.
- They actually focus on learning over grades
- They better understand the tradeoffs between studying and grades. Several make a conscious decision to earn the B to save themselves some time (at the expense of learning) but all of this is clear to both of us. THEY made the choice and they earned that grade.

Philip's Pre-GFE Experience

I retired prior to encountering GFE but I had actually had experience similar to GFE practices. I had developed course outcomes, used assessments with retakes, and planned teaching in a manner similar to GFE. The material below discusses these elements and provides some examples that might be useful to others considering GFE.

In 2012 Stephen Hughes and I began working together, but teaching separately, to improve our instruction. Our effort focused on Stephen's idea of opening a course by showing students a number of programming tasks that they should be able to solve by the end of the course. Though this was prior to GFE, we did employ several instructional elements used in GFE—1) course outcomes identification, 2) competency demonstration quizzes as assessments of programming fundamentals, and 3) assessment retakes.

My efforts were in a Visual Basic course that spent about half the course introducing programming fundamentals and half the course having students work on projects while addressing additional programming elements. The course was intended for non-majors and meant to produce professionals who could program, not professional programmers. The overall course goal was:

> Students are willing and able to use programming to solve a problem of interest to them.

Using backward design, we generated a goal decomposition, something like:

- Correctly produce simple programs involving programming fundamentals
    - Use Visual Studio to develop programs and submit them to the instructor
    - Solve simple IPO (input-process-output) coding tasks requiring input, output, and assignment; with assignment involving both numeric and string literals, variables, and common functions
    - Generate expressions involving Boolean variables, operators, and functions
    - Solve simple coding tasks that require selection (`if` statements)
    - Solve simple coding tasks that require repetition (`for`, `while` & `repeat` statements)
- Produce a program requiring the above programming fundamentals, modularization, and file I/O to solve a problem of the student's identification.

The programming fundamentals were assessed with "competency" demonstrations[1], timed paper and pencil quizzes (minutes) for which grades were recorded as + (competence evident), - (competence not evident), ? (can't tell, come see me). Students were allowed to retake competency demos. At first there were no restrictions on retakes but in later courses various time and frequency limits were tried. I would prefer no limits but the one making the most sense to me was that a retake be allowed up until the beginning of instruction on the unit after the next one if the outcomes depend on prior work. Copies of sample competency demos are available via the web

- https://www.cs.uni.edu/~east/teaching/vb/compDemos_current/I-sample.html
- https://www.cs.uni.edu/~east/teaching/vb/compDemos_current/II-sample.html
- https://www.cs.uni.edu/~east/teaching/vb/compDemos_current/III-sample.html
- https://www.cs.uni.edu/~east/teaching/vb/compDemos_current/IVsample.html
- https://www.cs.uni.edu/~east/teaching/vb/compDemos_current/Vsample.html

Instruction generally consisted of class demonstrations of new content, the assignment of practice/learning activity (homework), class Q&A and discussion of homework, and sharing of sample competency demos. The homework was quite extensive and meant as an indicator of

[1] The Visual Studio outcome was assessed by having students create and submit a Visual Studio project interface

what might be on the competency demos (i.e. functions not on the homework would not appear on the competency demo). Copies of the latest copies of the learning activities are available on the web:

- http://www.cs.uni.edu/~east/teaching/vb/spr18/unit-I_practice.html
- http://www.cs.uni.edu/~east/teaching/vb/spr18/unit-II_practice.html
- http://www.cs.uni.edu/~east/teaching/vb/spr18/unit-III_practice.html
- http://www.cs.uni.edu/~east/teaching/vb/spr18/unit-IV_practice.html
- http://www.cs.uni.edu/~east/teaching/vb/spr18/unit-V_practice.html

The combination of learning activities and assessments can, I believe, provide useful insight for those considering GFE.

The more general programming outcome was assessed with a programming project and the final exam. The final exam was a VB version of the benchmark exam (Simon et al, 2016) plus a "small" programming problem, e.g.:

> Read text from a file (words.txt, in the bin/debug folder) and produce the counts of the various word lengths (count all words of length 20 or greater as length 20). Report the counts by producing a string that will be displayed in a message box. Each count should be on a single line in the string you produce (i.e., concatenate VBNewline onto each line).

The benchmark exam provided a comparison that allowed for course evaluation. Unfortunately, I did not grade the final in a manner that could also have been considered an alternative assessment of the programming fundamentals. I'm not sure I would want to use it to totally replace the competency demos but can imagine it being used somehow to assess capability with programming fundamentals.

Our Reactions

In general, our experience with GFE has been extremely positive.

Grading has become simpler and easier as we no longer have to provide a numeric score on a wide range (1-10, 1-25, 1-100). We either don't grade or don't count most homework (which required adjusting to provide feedback in some other manner). Even though we now allow retakes on assessments, we actually spend less time grading, even if the time for developing alternative assessments is included. Since homework is not graded, cheating on homework is not a problem and we don't spend time or effort on countering it.

There is no arguing over points and students rarely question our assessments. Students appear happier with GFE than with our prior grading approaches. We believe they now think in terms of their own understanding of course content rather than grades or points.

The use of backward design and focus on outcomes has enhanced course planning. It is relatively easy to develop unit outlines and provide students with really effective study guides at the beginning of units to give them advance organizers. To some extent we are now "teaching to the test", but the tests are so general that doing so is good rather than bad.

Finally, we recently realized that the "equity" in Grading For Equity is more powerful than we had realized. The current attention on incorporating gender equity with its expanding scope, cultural equity with a host of student backgrounds, and being inclusive with respect to various student characteristics makes providing equitable instruction seem an overwhelming task. We recognized that we could achieve many, though perhaps not all, of the equity and inclusion goals by “simply” using GFE.

Recommendations

We would like to recommend that everyone redesign all their courses to apply GFE principles and practices. Unfortunately, if you are teaching more than one course that would likely be an insurmountable task—identifying assessable course outcomes is a truly significant endeavor. However, once that is done, incorporating GFE practices becomes relatively straightforward and probably not much more time consuming than regular class planning. So, if possible, we recommend taking one course and converting it to GFE.

If that isn’t possible, there may be something positive that is doable without changing a complete course. Feldman thinks so (see Stephens-Martinez, 2021). For courses that involve projects, preliminary content might be converted to GFE in a manner similar to what Philip did in Visual Basic. Some other possible piecemeal approaches include:

- Use a small scoring scale (i.e., 1-4 or less)

  As noted above a smaller grading scale can make grading easier. It seems a relatively simple process to make this change. When describing the grading scale be sure to describe what the score indicates or include a rubric.

- Avoid weighted averaging of category scores

  Totaling or averaging scores in various categories and weighting the category scores to produce an overall grade tells you almost nothing about how the student performed on anything. It is better to record scores on individual assessments and then base the overall grade on how many were completed at what level. This approach can even be applied on exams, particularly those that are reused. For example, each question could be graded using a four-point scale. If some items were more important, they could be counted as two or three items instead of one. The overall score on the exam would be based on the number of items completed in a satisfactory manner similar to how Ben graded PEEE.

- Not grading homework

  As noted above, not grading homework offers many advantages—perhaps most notably not worrying about cheating on homework. Homework should be for learning, not assessing learning. And, not grading homework lessens instructor time. Alternatives for providing student feedback need to be devised.

- Not grading behavior

  The big benefit of not grading behavior is the reduction of possibilities for bias in grading. It is relatively simple to accomplish, once you decide you want to do it.

- Allow retakes on some key elements of the course

  If you can’t imagine doing retakes for everything in the course, consider doing it for some of the material early in the course so students can overcome problems early in the course that might arise from getting used to college, the material, the professor, etc. An early F can keep a student from getting an A even if they eventually are getting all As in the course.

- Devising alternatives to grading group work products

  This has been one of the most difficult elements of GFE for us. If you can accomplish this you will have made a significant achievement. Possible alternatives might include assessing work individuals bring to the group, individual reflections of their skills, or individual reports and assessments of their group work and growth.

If you choose to implement GFE to some extent, we recommend telling the students that you are working to make grading more accurate, bias-resistant, and motivating and that there may be some hiccups in the process as you learn about it. Also, you should feel free to contact us. We want to help.

References

Andrew Berns. 2020. Scored out of 10: Experiences with Binary Grading Across the Curriculum. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 1152-1157.

Carol S. Dweck. 2006. *Mindset: The New Psychology of Success*. Random House, New York.

J. Philip East & Stephen Hughes. 2013. An experience report of our teaching Visual BASIC using a problem-oriented approach. *Proceedings of the 46th Midwest Instruction and Computing Symposium*, April, La Crosse, WI.

J. Philip East & J. Ben Schafer. 2005. In-person grading: An evaluative experiment. In *Proceedings of the Thirty-Sixth SIGCSE Technical Symposium on Computer Science Education*, 378-382.

J. Philip East. 2000. Teaching, assigning, and grading: Continuing efforts to enhance learning. *Proceedings of the 33nd Midwest Instruction and Computing Symposium* (formerly the Small College Computing Symposium), April 13-15, 2000. St. Paul, Minnesota. (available from the author)

J. Philip East. 1998. Including quality assessment in programming instruction. Presentation at *NECC '98*. June, San Diego, California.

Joe Feldman. 2019. *Grading For Equity: What It Is, Why It Matters, and How It Can Transform Schools and Classes.* Corwin, Thousand Oaks, CA.

Simon, Judy Sheard, Daryl D'Souza, Peter Klemperer, Leo Porter, Juha Sorva, Martijn Stegeman, Daniel Zingara. 2016. In *ICER '16: Proceedings of the 2016 ACM Conference on International Computing Education Research*, 103-111.

Kristin Stephens-Martinez. 2021. Grading For Equity with Joe Feldman. *CS-Ed Podcast, Season 2, Episode 4*. Retrieved from https://csedpodcast.org/.

Grant Wiggins and Jay McTighe. 2005. *Understanding by Design (Expanded 2nd Ed.)*. Association for Supervision and Curriculum Development, Alexandria, Virginia.