# Extreme Ungrading: Rewilding the Classroom through Human-Centered Design

Johanna Brewer
jbrewer@smith.edu
Smith College
Northampton, MA, USA

## ABSTRACT

Assessment in computer science education has grown reliant on rigid rubrics and intensive exams, a practice that yields capable yet compliant coders. In this article, I explore how we might use human-centered design to reexamine contemporary pedagogy and redesign our classrooms to cultivate a different type of programmer, one with a more critically engaged eye. Inspired by the ethos of agile development, I offer an alternative evaluation paradigm: Extreme Ungrading. Exploring results of a two-year case study applying this method to a software engineering class, this article distills actionable guidelines for enhancing learning outcomes through inclusive course development, and seeks to spark debate about our duty as scholars of HCI to reshape computer science education.

## CCS CONCEPTS

• **Human-centered computing**; • **Social and professional topics → Computing education**;

## KEYWORDS

ungrading, inclusive pedagogy, human-centered design, computer education

**ACM Reference Format:**
Johanna Brewer. 2024. Extreme Ungrading: Rewilding the Classroom through Human-Centered Design. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems (CHI EA '24), May 11–16, 2024, Honolulu, HI, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3613905.3644048

## 1 INTRODUCTION

Computational technology has undeniably revolutionized human interaction; the world is now awash with the products of real-time remote collaboration. As startups bloom and FAANG corporations boom, newly graduated software engineers, AI researchers, UX designers, infosec officers, data scientists, and system administrators find themselves powering the 21st century's radical and rapid transformation. Though much has changed in society since I was a college student during the first dotcom bubble, computer science instruction has remained remarkably unaltered in the decades following that burst. Transparencies on overhead projectors have been replaced by wireless tablets projecting digitally, and communal SPARCstations clustered in basement labs have given way to laptops nestled in backpacks. Yet, just as the fundamentals of logic and control structures stayed the same, so too have the ways for assessing mastery of core computing concepts. In this article, I will explore how the lens of human-centered design might be used reflexively to reframe the pedagogical purpose of assessment, and to refocus attention on our interdisciplinary obligation to graduate young scholars who can not only competently create, but also carefully critique, the next generation of tech.

### 1.1 The Art of Assessment in Computer Science

Grading in the field of computer science has long relied on rubrics that break down the scores for programming assignments into myriad specific logical or syntactical requirements (with individual point values) in an effort to produce "objective" evaluations [1]. But in the late 20th century, as instructors found themselves lecturing to hundreds of students who were producing thousands of lines of code, many CS departments grappling with that growth attempted to maintain rigor in their grading practices by tasking student teaching assistants with assessing the work of their juniors according to those professor-defined rubrics [31]. Following its release in 1997, many grading workflows began to rely on Alex Aiken's Moss (Measure Of Software Similarity) system for detecting plagiarism; in short order computer-assisted evaluation of student code became the norm [9], and scholars began to pursue the development of automated grading tools as an efficient and equitable approach to the scalability issues they were facing [13, 14].

The impulse to thoroughly quantify the performance of CS students stems not least from a love of decomposing problems into fundamental, modular components. Coders are taught to separate their concerns and so it seems logical in a discipline owing its existence to the binary system that professors would devise a bitwise approach to assessment. Yet, technologists also celebrate intuitive interfaces, algorithmic creativity, and elegant code, qualities of excellent programming which remain difficult to define in a rubric, let alone grade automatically. Any given framework of evaluation will produce different results, and as its effects compound over time, a particular sort of code, and coder, will emerge as the optimal outcome of the system. If we assert that objective assessment of software is not only possible but preferable, and if we aim to build fully-automated massively-scalable scoring solutions to render those judgements, what sort of developers do we expect such a learning process to produce? *How might we use a human-centered approach to reexamine contemporary pedagogy and redesign our classrooms to cultivate a different type of coder?*

### 1.2 Graduating into the Automation Generation

As it stands today, only 21% of computer science bachelor's degrees are awarded to women [16] and just 28% of Silicon Valley professionals are female; Latinx (5%) and Black (3%) folks are similarly underrepresented in the workforce, with numbers of Native American and non-binary coders so small they are often unreported [37]. The discipline is described by many students as unwelcoming, and though hiring bias affects all industries, it seems rather insidious when big tech companies publicize flashy diversity-building initiatives, even soliciting recruits to build software that will supposedly help eliminate bias, while perpetuating the problem through their own internal talent acquisition practices [11].

Tech has earned a reputation for being toxic, thanks in part to the grueling hazing-like hiring process of technical interviews popularized by big corporations, where candidates solve their way through an onslaught of standardized toy problems like a "gladiator fighting in the Colosseum for entertainment" [5]. This process, though purported by managers to be objective, meritocratic, and scalable, does not remotely resemble realistic scenarios that working developers encounter, and is frequently described by candidates as highly subjective, quite stressful, and distinctly non-inclusive.

On its face, striving for universal standardization, full automation, and reliable replication of code(r) evaluation sounds appropriate for a discipline rooted in logic, but pursued uncritically and exclusively, such intentions, noble as they may be, produce unfavorable and unsustainable outcomes. In a labor landscape where stable jobs of yore have been replaced by piecemeal precarious gig work [30], six-figure software design salaries represent one of the last attainable pathways to a better quality of life for marginalized students. But as the churning wheels of platform capitalism turn their strategies of disaggregation and depersonalization on the programmers it once prized, aspiring developers now find themselves contending with increasingly exploitative working conditions [32].

Academic information & computer scientists have both a unique opportunity and, I argue, an urgent obligation, to intervene in the calcification of this *compliant coder pipeline*. Rather than doubling down by teaching to the Leetcode test, instructors–especially those specialized in HCI–could do well to explore how the creative reimagining of our existing pedagogical systems might represent a viable means of reprogramming the status quo [10]. Indeed, this necessary reexamination and redesign of our culture of assessment has already begun; it can perhaps be best evidenced by the growing interest in *ungrading* as an alternative to traditional evaluation.

## 2 UNGRADING 101: A BRIEF SURVEY

Ungrading is not new, but it has been gaining popularity in higher education over the past five years after professors like Susan D. Blum [8] and Jesse Strommel [35] began speaking openly about their adoption of the practice and how it benefits learning outcomes. Though there is no formal definition, *ungrading* is generally understood as assessment that *minimizes (or eliminates) point-based scoring of student work in order to emphasize formative feedback*. Ungrading is an umbrella term for a pedagogical attitude that questions the contemporary function of assessment; it encompasses a range of practical approaches including: mastery grading, contract grading, standards-based grading, and specs grading.

Alternatives to the "traditional" quantitative assessment scales of 18th and 19th centuries have been in development for decades. Benjamin Bloom first introduced *mastery grading* in 1968, arguing against the acceptance of the bell curve by asserting that the overwhelming majority of students should be able to be achieve A-level mastery of the material given the right environment and sufficient time [7]. He championed the importance of defining clear specifications for learning objectives so that students themselves can comprehend if they have mastered the material. Linda Nilson built on this foundation by proposing *specifications grading*: a method of evaluation where students receive detailed *specs* for acceptable completion of assignments (typically to the B-level) and then complete modules (bundles of assignments) assessed Pass/Fail to demonstrate mastery of key learning concepts [28].

When implemented, these foundational forms of ungrading underscore not just the viability of assessment alternatives, but also the vitality they bring to the classroom. Seeing weaknesses in the constraining frame of 100-point grading, Joe Feldman shared an extensive discussion of how transitioning to *standards-based*, simple, 0-4 scale led to a more equitable learning environment [15]. Though focused on K-12 contexts, Feldman's work offers evidence that ungrading can increase inclusion, which is precisely what CS departments striving to close the representation gap hope to achieve. However, thanks to their collective tendency towards automation, the gap between traditional grading and ungrading may be wider in computer science & informatics than any other academic fields.

It is perhaps unsurprising, then, that much of the ungrading discourse in higher education has been driven by humanities and social sciences faculty who have years of experience iterating their techniques [18, 21, 23]. English professors, for example, describe effective methods for leveraging peer review to teach students to write for a broader audience than the course instructor–an approach that can usefully be adopted by other communication-focused disciplines. Yet, while code reviews and design crits have come to serve as important means of formative assessment in many CS classrooms, they cannot suffice for summative assessment when all the peers are novice "speakers" of the programming language or framework. Specialized techniques for adapting an ungrading approach to the rigors of the field are clearly required. Fortunately, we can look to recent reports from STEM instructors, especially mathematicians and chemists, who have provided inspiration by sharing candid reflections on their methods for ungrading in courses with significant technical content [12, 19, 22, 26, 33, 36]. These fields are also known for their rigid rubrics and grueling examinations, but these instructors report that relaxing their reliance on providing point-based assessments has helped reduce students' grade anxiety and increase their engagement with the challenging material.

### 2.1 Deprogramming in Progress

Despite the fact that computer scientists were the first to automate assessment, they have an equally long history of working on ways to shift students' mindsets away from accumulating or collecting points and towards mastering core concepts [39]. Mastery grading has been applied in computing classrooms since the 1990s, but most examples of the practice focused on introductory programming or courses for non-majors [17]. Indeed, when ungrading is applied in information & computer science, typically it is in the context of converting introductory courses from letter grades issued in the ABCDE/F format to binary Satisfactory/Unsatisfactory (Pass/Fail) scores in an effort to lower the barrier of entry to the discipline [25]. Though the avant-garde of the field may appear to be lagging behind the broader academic conversation, there is a significant cohort of computer scientists swapping stories about radically refactoring their approaches to assessment [24], including those who have been organizing a yearly hands-on workshop for introducing specifications grading to CS courses since 2019 [27].

Andrew Berns, for example, described reducing grading load while increasing comprehensive coaching through the development of a binary assignment grading system where students completed a defined number of activities Satisfactorily to earn a given letter grade for a course [6]. Karina Assiter similarly demonstrated how eliminating the grading of formative assessments and reducing the weighting of exams in favor of projects was able to increase retention of diverse learners in a variety of CS courses [2]. At the 2023 meeting of SIGCSE, Scott Spurlock shared how an approach that eliminated numeric grades, allowed resubmission of assignments, and encouraged students to give input on their final letter grade, served to improve overall student motivation towards tackling the challenging material of upper-level courses [34]. And at the same conference, Ella Tuson and Timothy Hickey presented their findings from experimenting with a mastery approach by applying specs grading to a software engineering course over 140 students [38]. In their semester-long trial, they too were able to reduce their grading burden while maintaining academic rigor, and plan to continue iterating their practice.

### 2.2 Addressing Alternative Assessments at CHI

There is clearly mounting evidence that moving beyond a piecewise, depersonalized, automated approach to assessment offers tangible benefits for students and instructors of computing alike. Yet there has been comparatively little discussion at CHI about the design of our courses and sparse debate about the responsibility we have as scholars of sociotechnical systems to contribute to the development of pedagogical best practices for computer science education. Rich Halstead-Nussloch and William Carpenter's work from CHI 2002 offers a rare example of a piece focused on instruction; in their short paper they describe applying a Bauhaus, studio-oriented, model to their classroom that encouraged the development of individual mastery in a collaborative team environment [20].

In a similarly brief 2017 CHI Note, Mihaela Vorvoreanu and co-authors present a focused case study that takes an integrated studio approach to UX education [40]. Their short paper was intended to define a space for pedagogical research at CHI, but it seems that objective remains unrealized. As we have seen, most research regarding computing pedagogy is shared at education-focused conferences. It is of course excellent to see innovative methods like GenderMag–a tool for teaching inclusive design to HCI students–being presented to the broader CS audience at ICER [29]. But with this piece, I want to spark discussion amongst the vanguard of our community, to unpack ungrading at alt.chi, and debate our collective duty to reshape computer science classrooms.

## 3 EXPERIMENTING WITH EXTREMES

Contemporary technology is rapidly reshaping our lives; today's coders must confront greater ethical, economic, and environmental challenges than ever before. It is becoming clear that by shifting the emphasis of assessment beyond syntax and structure, we can hone a more holistic approach towards computing education–one that develops skills like critical thinking, clear communication, effective collaboration, imaginative problem-solving, and personal responsibility. I would like to suggest that by leaning into this human-centered pedagogy we can create more inclusive learning environments that can be fertile grounds for planting seeds of positive social change.

As a member of the faculty at Smith–a historically women's liberal arts college–I have the privilege to teach at an institution that has been on a mission to shift the balance of representation since its inception. By 2021 when I joined, the computer science department had already made great strides in attracting a diverse group of students; that year Black, Hispanic, and Native American scholars made up 15% of our majors, 14% were Asian American, 6% multiracial, 36% of Smithies in CS were international students. 100% of those students were women, trans, and/or non-binary.

Yet even in an inclusive environment where the demographics represent an impressive outlier for the discipline, the legacy of traditional assessment can be felt. Students at a selective school like Smith suffer higher levels of mental health challenges, especially those related to anxiety. Perfectionism and an obsession with straight As run rampant on campus. As professors struggle to keep grade inflation in check, we are often confronted by students experiencing real crisis over minor flaws with their academic performance. When students are drilled with notion that every point counts, and life-changing job offers are the winning prize for maximizing their score, the bleak ramifications this unintentional gamification become apparent.

Understanding the stakes after a decade spent as a startup CTO, I am motivated to explore evaluation methods might be evolved to better prepare students for the complexities that computer scientists now face. Tasked with teaching human-computer interaction and software engineering, I naturally planned to introduce students to the Agile Manifesto and tenets of Extreme Programming [3, 4], but I was also inspired by the crucial influence these frameworks have exerted by distilling quietly radical principles from the cutting-edge practices of their peers. Accordingly, I would like to offer for discussion my own parallel objectives–a set of mandates reimagining the purpose of pedagogy to meet the moment of computational revolution we find ourselves in.

***Extreme Ungrading is a commitment to:***

❶ *Prepare students to thrive in industry or academia when they face complex conflicts without clear guidelines to follow;*
❷ *Teach students how to self-direct their learning through fast failure and not get hung up on appearing perfect;*
❸ *Orient students towards self-improvement and collective impact by prioritizing demonstrable positive results over points;*
❹ *Foster collaboration between students with diverse skillsets & backgrounds in a comfortable, yet challenging, environment;*
❺ *And, ideally, train students to accurately assess their work and others' based on observable outcome rather than effort.*

Following the fiat of this nascent framework, I have engaged in a two-year experiment to rewild the CS classrooms at Smith College. In the rest of this article I will explore the implementation of these pedagogical principles by presenting an in-depth case study; over the course of four semesters I attempted to validate this *Extreme Ungrading* orientation by applying a human-centered approach to the (re)design of an undergraduate software engineering class. This piece explicates that iterative process and, by reflecting on the outcomes, derives actionable guidelines for course development.

## 4 SOFTWARE ENGINEERING: A CASE STUDY

To encourage discourse about ungrading in the CHI community, herein I describe how my effort to design, test, and refine alternative assessment techniques, in tandem with supportive curricular structures, unfolded over four semesters, from Fall 2021 to Spring 2023. In the coming pages, I detail my exploratory experience guiding 113 students (avg. 28.5/semester) to mastering software engineering fundamentals by eschewing exams and rigid rubrics in favor of substantial assignments and scaffolded self-reflections.

Though I was certain after a decade away in industry that I wanted to engage in ungrading upon my return to academia, I was rather unsure of precisely what that would entail in practice. Thus, my initial approach to ungrading was wildly experimental, planned on the fly, and somehow both confidently bold and vaguely indecisive. Candidly, when I began this study of Extreme Ungrading, I was not quite sure how it should work, or even if it could work. Yet by committing to an agile process of inclusive student-centered curriculum design, I have developed a well-received course that enables first-time full-stack developers to achieve outcomes which continually exceed my expectations.

Following a participatory approach to the course design, I incorporated student perspectives from the start. Rather than implementing classroom policies that are often syllabi boilerplate, I took cues from universal design to maximize the overlap with the most often used academic accommodations. Before the course began, I circulated an intake survey; and class concluded with a detailed retrospective survey in addition to the college's formal course feedback form. Throughout the semester I solicited feedback from students through in-class polls, chat server discussions, and office hours. I took copious notes about student performance during and after classes; and I kept a running log of changes I planned to make for the subsequent semester as the course unfolded. Finally, I sought input from four colleagues who observed my pedagogical practice on three occasions.

### 4.1 First Run "Into the Wilds" (Fall 2021)

My first semester at Smith, I had the opportunity to redesign our software engineering course. Mastering this branch of CS means moving beyond solving single problems with short programs, to building systems tackling novel issues with thousands of lines of code, running on globally distributed machines, connecting potentially billions of users. Such a massive jump in scale necessitates learning new techniques to design applications that can be built by a team. The hard part of software engineering is drawing a reasonable map to success that a group of developers can execute together, so giving students a pre-determined rubric of todos would only undermine their ability to acquire those skills.

My aim with this updated course was to simulate the experience of a junior developer who joins a new team building a speculative product in an unfamiliar framework. Students begin by setting up their individual development environments to get familiar with the Ruby on Rails stack, then shift to working in teams of 4-6 to specify, plan, and build a unique bike sharing service designed to serve the community of a fictional region, Nipmuc Notch, resembling the area around our campus. The class is meant to prepare students to collaboratively learn on the job, cope with the inherent uncertainty of engineering, and deliver functional software ready to be deployed "in the wild" while working in messy conditions under realistic constraints. My motivation for ungrading was to inspire students to earnestly try (and fail) to put software engineering techniques into practice "for real" to produce tangible outcomes, rather than earn points by jumping through well-defined hoops of implementation.

It was this intention I attempted to communicate from the start. Before the first meeting I had students complete a survey rating their skill level on a variety of software engineering topics (e.g. agile development, pair programming, wireframing, version control) to establish a baseline of their abilities. Evaluation forms like these took the place of exams, and I made clear we would conduct self and peer reviews throughout the semester. I explained each weekly 2.5 hour session would follow a workshop format, beginning with a short lecture to unpack the readings, then switching to collaborative hands-on activities putting those learnings into practice.

Given that building a working software system was their overall goal, I stated that assessment would be weighted: 50% major project; 25% reflections & report; and 25% participation. I explained I would provide assessments for their work on a *simple scale* of: Needs Improvement; Meets Expectations; Exceeds Expectations; and Distinguished. Stressing I could only differentiate software at granular levels, I indicated this scale loosely mapped to C at the low-end and A/A+ at the high-end, and repeatedly assured students that "anyone who Meets Expectations will earn a B or better."

During the course's first half, students focused on acquiring foundational knowledge and finding teammates. They were assigned two chapters each week, given optional technical trainings, and required to submit 200-300 word reading reflections to prepare for in-class activities. In the first three weeks, I intentionally shuffled students into different work groups and organized sharing circles, giving them ample structured opportunities to meet. Students then filled out a group preferences survey to describe their working style and skills, indicate which classmates they would like to collaborate with, and mention anyone with whom they could not be productive. Using their stated predilections, my own observations, and the results of their first local stack configuration assignment, I sorted the students into scrum teams of 5-6 developers who would work together for the remaining 10 weeks of the semester. My strategy was to match students with compatible skill levels and social affinity into productive "rowboats" that could pull in harmony.

Group projects often make students shudder, but when teams were announced and sent on their first mission (to observe a competitor's bike share system), there was enthusiasm in the air. On the other hand, the reading reflections produced much more anxiety despite being low-stakes assignments where nearly everyone Met

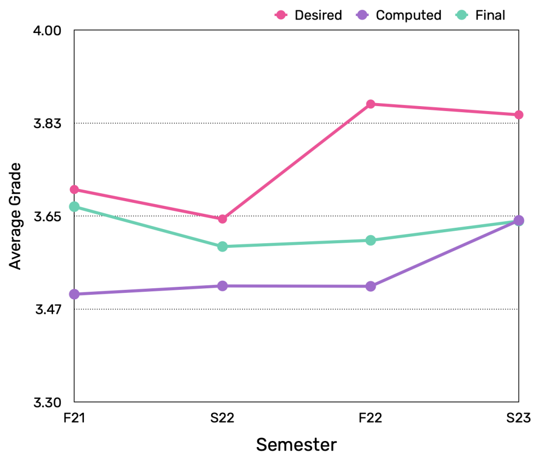

**Figure 1: Comparison between students' average *desired* and actual *final* grades across four semesters, with normalized *computed* scores for reference.**

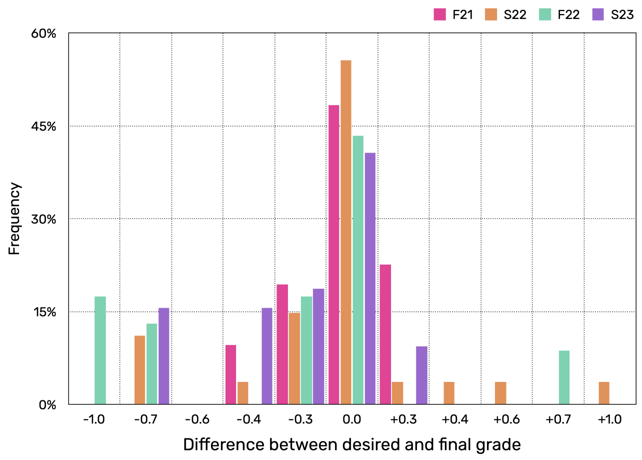

**Figure 2: Distribution of deltas between students' *desired* and assigned *final* grades, compared across four semesters. Negative values indicate lower final grades than desired, positive values correspond to final grades higher than requested.**

Expectations. Some worried students asked how submissions that Exceeded Expectations could become Distinguished, and finding myself struggling to assess that ineffable difference, to ease the tension I informed them mid-semester they could resubmit any past reflections. Nonetheless, the amount of emotional energy we spent on those short writing exercises made it clear my methodology required further refinement to vent pointless pressure.

Fortunately, the success of ungrading the major project was far more definitive. I broke the group effort to develop a Ruby on Rails bike sharing application into four parts. Each assignment included a list of specifications for a deliverable (typically a mix of code, documentation, and retrospective) that would Meet Expectations, but there were no guidelines for the higher tiers. Students received detailed written feedback on their submissions, and clearly focused on that qualitative criticism. Many enthusiastically attended my office hours, packing into the room with questions about improving their work as they advanced through phases of the project.

Yet, as the students completed their prototypes they seemed unsatisfied with their progress. Ruminating, I resolved to commit more decisively to ungrading, and announced that their final retrospective assignment would ask for input on their letter grades. Though I would reserve the right to determine grades as I saw fit, I informed students I would take their self-assessments seriously if they made a clear case about performance with respect to the guidelines laid out in the syllabus. This late-game decision was partially a way to deal with the creeping anxiety around reading reflections, but mostly it was to show the students that I truly meant it when I said my primary concern was about developing their senses of self-awareness, creativity, and resilience as software engineers.

All of my emphasis on embracing and pushing through failure to produce tangible results seemed to really pay off. As teams shared their Minimum Viable Products (MVPs) during our final class demo day, there was a mutual understanding among peers about whose software features were most impressive and *why*. Exiting the room there was a palpable energy of shared respect for what they had all achieved: designing and building a real piece of functional software from nothing–with some even deploying it on the world wide web.

Given that asking for student input on grades was a surprise twist, I needed to fold a new layer into my nascent ungrading system. When it came time to submit final evaluations, I approached the matter as a fitting problem. First, I translated my simple scale into grade points, weighted the assignment scores, and calculated the *computed grade* for each student. Next, I compared that score to the *desired grade* submitted in their self-assessment and examined the delta. Taking the justifications about their learning efforts and outcomes into account, and reflecting on their performance over the semester, I then assigned a *final letter grade* for each student (see Figure 1).

Receiving no change requests or complaints, it seemed students found the assessments fair. I agreed, judging them "more fair" than the computed or desired grades alone. Though I had sometimes struggled to evaluate students' work in isolation, reading the self-assessments offered deeper insight on their grasp of the material. While in some cases a student's overly inflated sense of accomplishment was evidenced by thin justifications of progress, in several instances students had actually undervalued their substantial achievements. Yet on the whole, self-assessments tended to concur with mine: 48% of students desired the final grade they got; 29% had asked for a higher final grade than they received; and 23% had requested a lower grade than what they ultimately earned. The adjustments I made were minor, redistributing slightly over- or under- confident students by just 0.3-0.4 grade points (see Figure 2).

Perhaps more important was their feedback. Students offered final reflections that spoke directly to my pedagogical goals, describing the class as a *"welcoming, effective learning environment"* where they had a *"safe, positive experience working in a software engineering team."* Many were empowered by their accomplishments, feeling *"fulfilled and validated applying my skills in a real world setting"* in a course *"as close as you'll get to a software engineering internship without an actual job."* My wild swings were connecting, and so despite my extremely experimental approach to ungrading, I finished the first semester unexpectedly assured by the outcomes.

## 4.2 Second Iteration (Spring 2022)

Reflecting on my ad hoc experiment, for the second semester I endeavored to establish the ungrading setup clearly from the start. Students were informed on the first day that they would give input on their overall letter grade in their final self-assessment. And in addition to the in-class and assignment retrospectives of the previous iteration, I also added a substantial midway self-review to give students the opportunity to contemplate and course correct.

Assignments and assessments followed the same arc as the prior term, but I shored up the structure. First, I provided a mapping between my simple scale and traditional letter grades: Needs Improvement (C) | Meets Expectations (B) | Exceeds Expectations (B+/A-) | Distinguished (A/A+). Then to further reduce grade anxiety, I stressed reflections were meant to be low-stakes, included new prompts in the instructions encouraging open-ended responses, emphasized to not sweat the assessment, but instead focus on engaging with the readings. For all individual assignments I adopted a clear *late is better than never* policy, allowing students to submit any solo work up until the last day of class for full credit. Finally, I made the three Ruby on Rails framework training modules mandatory assignments, only assessed on a Complete/Incomplete basis.

Adding these guideposts resulted in several positive changes. Anxiety over assessment of individual assignments was refreshingly reduced. Only a few students inquired about improving, and I overheard peers assuring each other not to worry about Distinguishing themselves since it was not a big deal. The quality of reflections was unchanged, but I agonized less over scoring them and completed my evaluations faster, allowing time for deeper feedback on the major project. Though there was technically more assigned work, students had fewer complaints about the load. More explicit scaffolding and expectation setting sprinkled throughout the course (like weekly timeline reminders, hourly targets for assignments, and code speed estimation guidelines) resulted in lower stress levels for students and myself. This improvement, in hindsight, seemed to reflect the basics of technical team management. Providing clearer policies while shifting responsibility and agency to the students ultimately resulted in a better experience on both sides of the lectern.

While the improvements were notable, I do not want to give the false impression that effective ungrading eliminates student anxiety. Rather, I suggest it has the potential to redirect fears towards more productive ends. Students came to my office hours anxious about the right things: that they were not grasping the material, that they were letting their teammates down because they lacked skills or confidence, that they were not managing their time properly. We developed strategies for tackling their troubles together and then they successfully implemented them on their own.

Teams took our final demo day even more seriously, especially given that I raised the stakes by announcing they would choose a winner. One group made entrepreneurial use of campus resources to print stickers for their service and every team seemed to put extra polish into their pitch. As I tallied their votes for the Most Valuable Minimum Viable Product (MVMVP), it was clear that they had developed the discerning eyes of well-trained software engineers.

Seeing students accurately evaluate and appreciate their peers' work was an excellent outcome. Curiously though, this cohort was both more accurate and generous assessing their own performance: 56% of students desired the final grade they got; 30% had requested a higher grade; and 15% under-valued themselves. Though the percentage of students whose assessments matched mine increased, the variance of those who missed the mark also grew. Several students' desired scores varied from their final ones by 0.6 or 0.7 points, and one student under-sold themselves by an entire letter grade. Despite needing to make such significant adjustments in a few cases, I once again received no complaints, bolstering my confidence that the ungrading methodology was, in fact, producing reasonable results.

## 4.3 Third Trial (Fall 2022)

Going into the third semester, I resolved to further hone the ungrading methodology by focusing my attention on improving the overall quality of student work. In an effort to increase interactivity during our weekly reading recaps, I reduced the required length of reflections to 100 words, and shifted to having students share them semi-publicly in our class Discord (chat server). Instead of assessing their quality, I only marked reflections for completeness, and used their thoughts to spark deeper discussion during lecture by breaking the ice through asynchronous participation.

To foster further support for reflection, I required short retrospectives with the training assignments, organized additional in-class code reviews, and expanded the midway self-reviews. In those, I asked students for a simple scale self-assessment, target final grade, and resolution for changes necessary to achieve that goal. The increased emphasis on self-examination and peer review led to more meaningful engagement with the material. By the time the students voted theirt MVMVP on demo day, I was again thoroughly impressed by their achievements. Several groups had implemented more robust, feature-rich bike sharing apps than those before, and I was thrilled to witness this new level of complexity.

Wanting to continue providing ample feedback on the final projects, while not getting bogged down in the details assessing their relative merits, I opted to create a thresholded checklist to streamline the quantitative component of my evaluation. Relying on the outcomes of the previous year's projects, I developed a set of clear criteria that would be marked as either fulfilled or not, and outlined tiers of completeness corresponding to my simple scale. This rarefied rubric allowed me account for the more outstanding applications produced that semester, while ensuring I could focus on giving all the teams a comprehensive review.

Surprisingly, as the quality of their work increased, so too did the generosity of their self-assessments. In this permutation of the course: 44% of students received the final grade they desired; 47% over-estimated their abilities; and 9% underrated their skills. Given that these students represent the pandemic microgeneration, it is possible the miscalibration between effort and outcome stems from spending the first years of college in the vacuum of Zoom. Yet again, I received no complaints about the adjustments I made, though one student, acknowledging that their group project participation had not been ideal, asked how they could have earned a higher mark. When I reiterated that their communication skills, key to software engineering, needed work, the next semester they came to my office hours for tips to improve. Overall, I was pleased with the enhanced caliber of the final projects, and proud to see what the students could do when their minds were freed to focus on what matters.

### 4.4 Fourth Refinement (Spring 2023)

Entering the fourth round of my experiment with far more certainty than I began, I concentrated on raising the quality of student work further, this time adding more interactivity and evaluation signposting. By updating two early in-class exercises, I created more opportunities for communication and collective skill building, which in turn improved overall camaraderie. Perhaps more significantly, by including new assessment overviews for assignments that defined the criteria for Meets, Exceeds, and Distinguished submissions, student anxiety around grades all but vanished. Apprehension around evaluation was fully eclipsed by a healthy sense of collaborative competition to achieve mastery of the material.

As teams reviewed each other's prototypes or gathered around for my seasoned consultations on their code, the atmosphere of peer support was palpable. And by the end of the term, the students had once again bested their predecessors, producing several final projects which exceeded the benchmarks of the previous semester's rubric. Here too as their capacity for growth heightened, so to did their sense of accomplishment: while 41% of students got the final grade they desired; 50% had asked for a higher one; and 9% undersold their achievements. Though I remain unsure about the source of this over-correction, there were again no objections to my adjustments, and overall even the students who had found themselves struggling with the challenges of the course ultimately had far better outcomes than before. As one explained: *"The hard times I had with my team were not failures, they were the best way to learn invaluable lessons about teamwork. I'm a better programmer now, knowing software engineering isn't easy, the processes are frustrating and painful, but to me that is the best part!"*

Taken to its extreme over the course of four semesters, my ungrading methodology achieved many of the effects I envisioned. In the absence in clear directions, while working through serious communication challenges, this student learned to embrace failure, and helped pull their team out of a nose dive to Meet Expectations by demo day. Seeing a wide array of junior software engineers build up resilience to the doubts and difficulties of the discipline, while delivering delightful products with deep values, demonstrates to me that Extreme Ungrading can indeed cultivate a different kind of coder, one that I would like to have more of in this world.

## 5 EXTREME UNGRADING ELEMENTALS

Hopefully this two-year case study will serve to spark further discussion about how human-centered design can encourage us to eschew our automated ways. In addition to the intermediate level software engineering course, I have also applied this methodology to an introductory programming class and an upper level human-computer interaction seminar. Drawing on all those experiences, I distilled four key elements of my Extreme Ungrading practice that seemed most crucial to the student successes I observed.

❖ *Simplified, straightforward, safe assessments.* Shifting focus from point-based evaluations towards qualitative feedback was the most challenging and rewarding aspect of ungrading. Avoiding a piecewise approach, by opting for a holistic tiered system with a *simple scale*, allowed the students to focus on achieving tangible outcomes instead of nickle and diming over minutiae. Though freedom from restrictive rubrics is a fundamental feature of ungrading, it was still essential to provide students with a sense of guided security, so they would feel safe to fail. Assessing low-stakes assignments for completeness alone, and shifting the purpose of such work towards enhancing classroom interactions, opened space for the trial and error necessary for collaborative complex skill building. Offering well-defined tiers for assessment for all the assignments made it straightforward for everyone to understand when the bar had been cleared. By ensuring that Meeting Expectations would earn a B, while offering a forthright but formidable path for those seeking to Distinguish themselves with an A, I crafted a classroom that had both an accessible floor and a vaulted ceiling.

❖ *Real-world deadline structure.* Time management is critical to software engineering, but my approach to teaching punctuality was not punitive. Deadlines for all assignments fell on midnights before class; and those due dates were laid out in a calendar revisited every week. Giving students reasonable structure and clear cadence modeled how they should pace themselves through a complex project. Providing fixed deadlines for group deliverables encouraged students to develop more self-awareness and personal responsibility in their workload management. When showing up prepared became a form of showing up for others, when class sessions were devoted to demonstrating progress and helping peers improve, students experienced how shared deadlines functioned as collective goalposts. At the same time, adopting a *better late than never* policy permitting students to submit any individual assignments before the last day of class (without penalty except delayed feedback) provided space and incentive to master material they could not complete on their first try alone. These twin policies made deadlines meaningful so students held themselves accountable accordingly.

❖ *Scaffolded, ceaseless self-reflection.* Retrospectives may be second nature for most agile developers, but regular intentional self-scrutiny is not something that most computer science students are yet in the habit of doing. Beginning in class, and continuing with every assignment, requiring brief *good thing, bad thing, better thing* retrospectives assessing personal progress constructively constrained students to keep track of their own growth. Charting their advances through a triptych of deeper self-evaluations (including a pre-class skill survey, a midway reflection, and a final self-team-peer review) and pairing those with the weekly retrospectives created a robust record of efforts and outcomes. Gaining such clarity on their learning process enabled both myself and the students themselves to more accurately assess the work they produced.

❖ *Communication-oriented classroom culture.* Acclimating computer scientists to constructive critique was a crucial component of my ungrading experiment. Offering qualitative evaluations on major assignments created a key channel for formative feedback, but it was merely one among many. By incentivizing students to use the chat server to reflect and pose questions, by praising peers who supported each other with advice, and by encouraging interactions with their Teaching Assistants, I purposefully knit social ties between the engineers so they could learn collaboratively. Strategic scrum team formation played a large role in the success of their cooperations. Rotating students through workgroups early in the semester and soliciting their preferences allowed me to place developers in productive pods. I continued that facilitation by helping groups abide by their team contracts while pairing them into different supergroups for code reviews. Underpinning all that organization was my own mentorship-driven workshop-oriented method of teaching. When working one-on-one, I engaged strictly in strong-style pair programming (never touching their keyboards, ensuring my ideas moved through their hands); I consistently encouraged students to attend office hours in groups and held regular open code consultations during class. Walking the floor and talking shop with the students reframed my role from austere overseer to approachable advocate; and opening the lines of peer communication supercharged their ability to master the complex material.

## 6 CONCLUSION

Extreme Ungrading as I have outlined it is not meant to be a rejection of traditional computer science rigor, rather it is posed as a necessary evolution of the way that we guide learners towards a mastery of our continually complexifying discipline. Automated point-based grading in academia is no more inevitable than the automation and quantification of gig economy apps that proliferate under platform capitalism. Yet unlike managers at massive tech congolmerates, members of the academy have the power to orient developing minds towards new horizons of our field, to inspire students entering the workforce to reimagine the role of technology in our society. By eliminating exams, rarefying rubrics, and prioritizing collaborative work products, it is possible to rewild computer science classrooms to become more inclusive spaces for exploration.

Scholars of human-computer interaction have the potential to reshape the future of the technology industry through the pedagogical precedents we set in the classroom today; my sincere hope is that more of us will begin to realize that. As to be expected with an agile approach, I plan to further refine my methodology in the ensuing semesters. By introducing formal verbal critiques for prototypes and MVPs, I will aim to improve student comprehension of their projects' merits, ultimately seeking to reduce the variance between their assessments and mine. I also intend to extend my Extreme Ungrading practices to upcoming courses in need of preparation, and I encourage readers to consider doing the same. While upending an existing system can be challenging, preparing new courses can be a perfect time to experiment with avant-garde styles of assessment. My hope is that this case study, and the practical recommendations I have derived from it, will provide enough supportive scaffolding for those tempted to take the same quantum leap; I look forward to reading the future reports of those blazing ever more rugged ungrading trails.

## ACKNOWLEDGMENTS

Many thanks are due to my brave, brilliant students for their willingness to explore the wilds of experimental course design, especially those in team Ghost who set the earliest records of achievement on untrodden ground, energizing me to continue.

## REFERENCES

[1] Tuukka Ahoniemi, Essi Lahtinen, and Tommi Reinikainen. 2008. Improving Pedagogical Feedback and Objective Grading. In *Proceedings of the 39th ACM Technical Symposium on Computer Science Education* (Portland, OR, USA) *(SIGCSE '08)*. ACM. https://doi.org/10.1145/1352135.1352162

[2] Karina V. Assiter. 2023. Integrating Grading for Equity Practices into Project-Based Computer Science Curriculum. *ACM Inroads* 14, 1 (2023), 22–29. https://doi.org/10.1145/3582559

[3] Kent Beck. 1999. *Extreme Programming Explained: Embrace Change.* Addison-Wesley.

[4] Kent Beck, James Grenning, Robert C. Martin, Mike Beedle, Jim Highsmith, Steve Mellor, Arie van Bennekum, Andrew Hunt, Ken Schwaber, Alistair Cockburn, Ron Jeffries, Jeff Sutherland, Ward Cunningham, Jon Kern, Dave Thomas, Martin Fowler, and Brian Marick. 2001. Manifesto for Agile Software Development. Agile Alliance. https://agilemanifesto.org.

[5] Mahnaz Behroozi, Chris Parnin, and Titus Barik. 2019. Hiring is Broken: What Do Developers Say About Technical Interviews?. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing* (Memphis, TN, USA) *(VL/HCC '19)*. IEEE. https://doi.org/10.1109/VLHCC.2019.8818836

[6] 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* (Portland, OR, USA) *(SIGCSE '20)*. ACM. https://doi.org/10.1145/3328778.3366956

[7] Benjamin S. Bloom. 1968. Learning for Mastery. *Evaluation Comment* 1, 2 (1968).

[8] Susan D. Blum. 2017. Ungrading. Inside Higher Ed. November 14, 2017. https://insidehighered.com/advice/2017/11/14/significant-learning-benefits-getting-rid-grades-essay.

[9] Kevin W. Bowyer and Lawrence O. Hall. 1999. Experience using "MOSS" to detect cheating on programming assignments. In *Proceedings of the 29th Annual Frontiers in Education Conference* (San Juan, Puerto Rico) *(FCE '99)*. IEEE. https://doi.org/10.1109/FIE.1999.840376

[10] Johanna Brewer. 2023. Seeing Like the Streamers: Reprogramming the Panopticon. In *Real Life in Real Time: Live Streaming Culture*, Johanna Brewer, Bo Ruberg, Amanda L. L. Cullen, and Christopher J. Persaud (Eds.). MIT Press.

[11] Phoebe K. Chua and Melissa Mazmanian. 2020. Are You One of Us? Current Hiring Practices Suggest the Potential for Class Biases in Large Tech Companies. *Proceedings on Human-Computer Interaction* 4, CSCW (2020). https://doi.org/10.1145/3415214

[12] David Clark. 2022. Assessing My First Semester of 'Ungrading'. EdSurge. January 17, 2022. https://edsurge.com/news/2022-01-17-assessing-my-first-semester-of-ungrading.

[13] Nathalia da Cruz Alves, Christiane Gresse von Wangenheim, Jean Carlo Rossa Hauck, and Adriano Ferreti Borgatto. 2020. A Large-Scale Evaluation of a Rubric for the Automatic Assessment of Algorithms and Programming Concepts. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (Portland, OR, USA) *(SIGCSE '20)*. ACM. https://doi.org/10.1145/3328778.3366840

[14] Emma Enström, Gunnar Kreitz, Fredrik Niemelä, Pehr Söderman, and Viggo Kann. 2011. Five Years With Kattis — Using an Automated Assessment System in Teaching. In *Proceedings of the 41st Frontiers in Education Conference* (Rapid City, SD, USA) *(FCE '11)*. IEEE. https://doi.org/10.1109/FIE.2011.6142931

[15] Joe Feldman. 2018. *Grading for Equity: What It Is, Why It Matters, and How It Can Transform Schools and Classrooms.* Corwin.

[16] National Center for Science and Engineering Statistics (NCES). 2023. Diversity and STEM: Women, Minorities, and Persons with Disabilities. Special Report NSF 23-315. https://ncses.nsf.gov/pubs/nsf23315/report.

[17] James Garner, Paul Denny, and Andrew Luxton-Reilly. 2019. Mastery Learning in Computer Science Education. In *Proceedings of the 21st Australasian Computing Education Conference* (Sydney, NSW, Australia) *(ACE '19)*. Association for Computing Machinery. https://doi.org/10.1145/3286960.3286965

[18] Elisabeth Gruner. 2022. I no longer grade my students' work – and I wish I had stopped sooner. Big Think. April 9, 2022. https://bigthink.com/thinking/ungrading.

[19] Beth Haas. 2021. Reflections on Ungrading in Small Chemistry Classes. Personal Blog. January 3, 2021. http://bethhaas.me/blog/2021/1/reflections-ungrading-small-chemistry-classes.

[20] Rich Halstead-Nussloch and William Carpenter. 2002. Teaching and Learning Ubiquitous CHI (UCHI) Design: Suggestions from the Bauhaus Model. In *Extended Abstracts of Conference on Human Factors in Computing Systems* (Minneapolis, Minnesota, USA) *(CHI EA '02)*. Association for Computing Machinery. https://doi.org/10.1145/506443.506533

[21] Monica Heilman. 2020. Ungrading Explained: What I'm Telling my Students This Fall. Personal Blog. August 25, 2020. http://monicaheilman.com/ungrading-explained.

[22] Claire L. Jarvis. 2020. Chemistry educators try 'ungrading' techniques to help students learn. C&EN. April 26, 2020. https://cen.acs.org/education/undergraduate-education/Chemistry-educators-try-ungradingtechniques-help/98/i16.

[23] Douglas King. 2022. A Defense of the "Ungrading" Movement. The James G. Martin Center for Academic Renewal. July 13, 2022. https://www.jamesgmartin.center/2022/07/a-defense-of-the-ungrading-movement.

[24] Aarti Madan, Geoff Pfeifer, Gillian Smith, Ryan Madan, Sarah Stanlick, and Zoe Reidinger. 2022. Food for Thought is Back! Save the Date for Ungrading the WPI Experience: Lessons from Across the Disciplines. Online. March 21, 2022. https://wpi.edu/news/announcements/food-thought-back-save-date-ungrading-wpi-experience-lessons-across-disciplines.

[25] David J. Malan. 2021. Toward an Ungraded CS50. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21)*. ACM. https://doi.org/10.1145/3408877.3432461

[26] Mariam F. Mashregi. 2021. Is Ungrading an Appropriate Assessment Tool in Science-Based Courses? Open Library. June 2021. https://ecampusontario.pressbooks.pub/tlhe720assessment/chapter/is-ungrading-an-appropriate-assessment-tool-in-science-based-courses.

[27] James W. McGuffee, David L. Largent, and Christian Roberson. 2019. Transform Your Computer Science Course with Specifications Grading. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery. https://doi.org/10.1145/3287324.3287528

[28] Linda B. Nilson. 2014. *Specifications Grading Restoring Rigor, Motivating Students, and Saving Faculty Time*. Routledge.

[29] Alannah Oleson, Christopher Mendez, Zoe Steine-Hanson, Claudia Hilderbrand, Christopher Perdriau, Margaret Burnett, and Amy J. Ko. 2018. Pedagogical Content Knowledge for Teaching Inclusive Design. In *Proceedings of the 2018 ACM Conference on International Computing Education Research* (Espoo, Finland) *(ICER '18)*. Association for Computing Machinery. https://doi.org/10.1145/3230977.3230998

[30] Alexandrea Ravenelle. 2019. *Hustle and Gig: Struggling and Surviving in the Sharing Economy*. University of California Press.

[31] Stuart Reges. 2003. Using Undergraduates as Teaching Assistants at a State University. In *Proceedings of the 34th ACM Technical Symposium on Computer Science Education* (Reno, Nevada, USA) *(SIGCSE '03)*. Association for Computing Machinery. https://doi.org/10.1145/611892.611943

[32] Juliana Feliciano Reyes. 2019. Will Google's Struggle With its 'Underclass' Lead to White-collar Workers Becoming the Next Labor Activists? The Philadelphia Inquirer. May 29, 2019. https://inquirer.com/news/google-temp-workers-labor-organizing-activism-20190529.html.

[33] Clarissa Sorensen-Unruh. 2020. Ungrading: What is it and why should we use it? ChemEd X. January 14, 2020. https://chemedx.org/blog/ungrading-what-it-and-why-should-we-use-it.

[34] Scott Spurlock. 2023. Improving Student Motivation by Ungrading. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education* (Toronto, ON, Canada) *(SIGCSE '23)*. ACM. https://doi.org/10.1145/3545945.3569747

[35] Jesse Strommel. 2018. How to Ungrade. Personal Blog. March 11, 2018. https://www.jessestommel.com/how-to-ungrade.

[36] Robert Talbert. 2022. Ungrading after 11 weeks. Personal Blog. March 20, 2022. https://rtalbert.org/ungrading-after-11-weeks.

[37] Donald Tomaskovic-Devey and JooHee Han. 2018. Is Silicon Valley Tech Diversity Possible Now? Center for Employment Equity at UMass Amherst. July 1, 2018. https://www.umass.edu/employmentequity/silicon-valley-tech-diversity-possible-now-0.

[38] Ella Tuson and Timothy Hickey. 2023. Mastery Learning with Specs Grading for Programming Courses. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education* (Toronto, ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery. https://doi.org/10.1145/3545945.3569853

[39] Mark Urban-Lurain and Donald J. Weinshank. 1999. "I Do and I Understand": Mastery Model Learning for a Large Non-Major Course. In *Proceedings of the 30th SIGCSE Technical Symposium on Computer Science Education* (New Orleans, Louisiana, USA) *(SIGCSE '99)*. Association for Computing Machinery. https://doi.org/10.1145/299649.299738

[40] Mihaela Vorvoreanu, Colin M. Gray, Paul Parsons, and Nancy Rasche. 2017. Advancing UX Education: A Model for Integrated Studio Pedagogy. In *Proceedings of Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery. https://doi.org/10.1145/3025453.3025726