4

Information Hazing

An Examination through Computer Science Education

Elizabeth Wickes and Melissa G. Ocepek

INFORMATION HAZING

Misinformation is experienced in many places and forms throughout a person's lifetime. Unfortunately, one common place where misinformation can have a long and lasting impact is within the classroom. As instructors working within the field of information science, we have seen students struggle with understanding the norms and rules of academia. Inadequate or incorrect guidance can cause negative impacts for students. These include missed opportunities, peer ostracism, anxiety, and lowered grades. These have long- term implications in terms of the students lived experience as well as larger societal representation in computing fields.

We define information hazing as the use of information to directly and indirectly harass and/or exclude newcomers. It is often used to create strong bonds of in and out groups in social environments that have shared values and social cohesion. It is used to demonstrate one's current membership within a group, or worthiness for gaining membership. This concept derives from the research literature on hazing in academic and work settings as well as research on bullying in higher education.

The concept of information hazing provides a new framework to consider how the norms of early computing education encourage malinformation and misinformation to spread and privilege students who are more familiar with computer science concepts, culture, and industries.

Most traditional examples of hazing showcase overt and direct behaviors between individuals. Information hazing is different in several distinct areas. First, it is often a covert approach, residing within policies or other governance structures of the social group. This content will often blend into surrounding policies, hiding behind other discussions of ethics and morality. Second, being embedded within governance and decision-making powers, it acts at scale on newcomers rather than individual actors. Third, these two factors intersect to obfuscate an individual person's ownership or connection to these governance structures. For example, the justification for a decision would come from the system rather than a person. Fourth, this

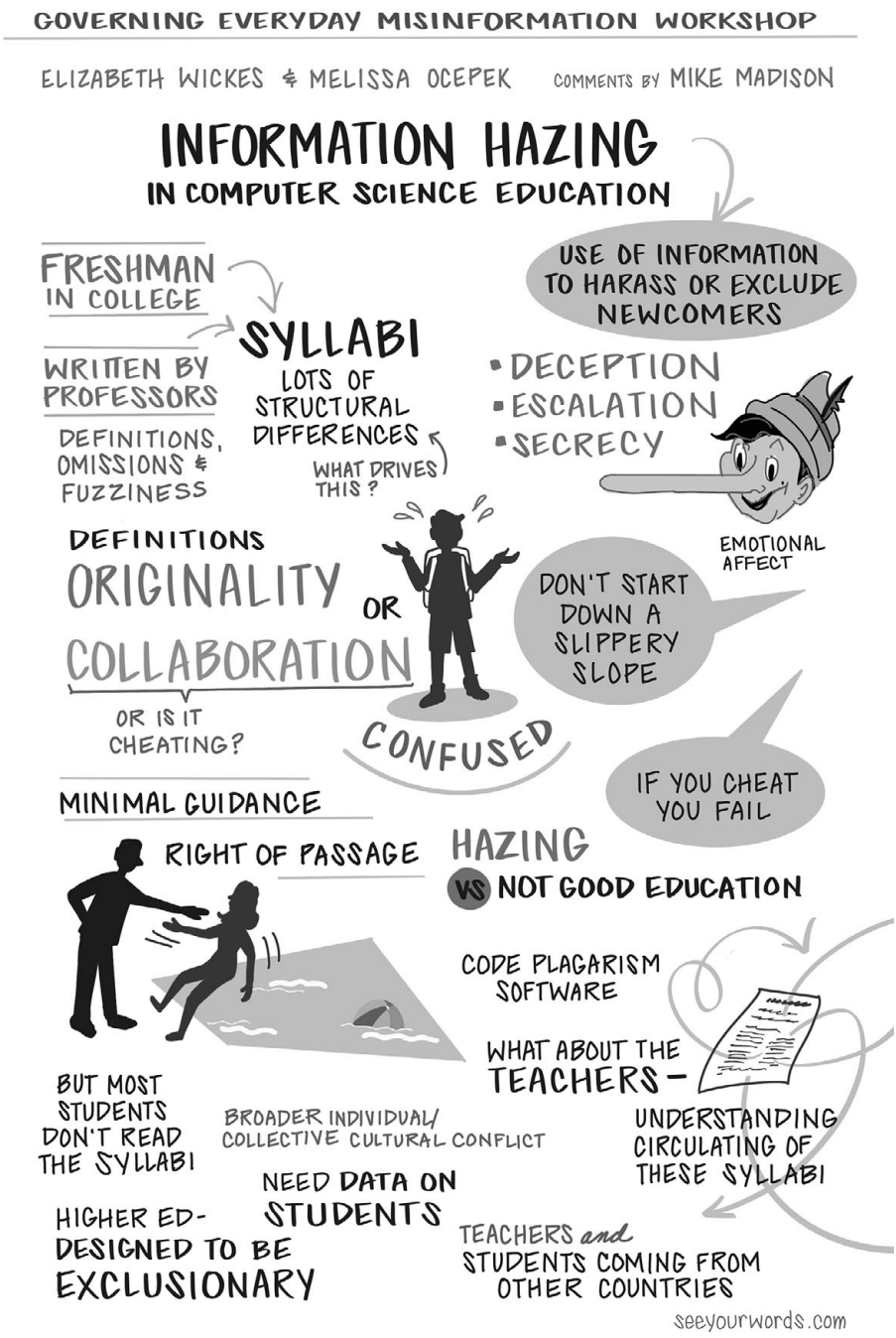

FIGURE 4.1 Visual themes from information hazing: an examination through computer science education.

independence allows the systems to operate outside the normal barriers of time. Policy language and precedents can last beyond the original author's lifespan or tenure within the group, with authorship credit often forgotten and attributed just to group norms.

These factors combine to make intersectional traps that further advance information hazing's ability to exist under the normal checks and balances designed to dismantle inequitable and unfair practices. While scholars have addressed many elements of computing education that have made it unwelcoming to many, no studies have considered how the governing documents of class including the syllabus, honor code, student code, and other academic integrity related policies begin the process of creating an unwelcoming environment when they are shared before or on the first day of class. These policies have been studied under the lenses of professional practice, detection, and prevention; there appears to be a gap of investigation into the more holistic impact of the policies on the student experience. In this chapter, we examine and consider the governing documents of first computing courses with an eye to information hazing. A form of hazing that we believe is all too prevalent in computing education and is a factor in the reason many nontraditional computer science students struggle with their grades, mental health, and long-term prospects in the field.

Hazing

As long as higher education has existed so has hazing. Plato described what modern-day scholars consider hazing at his Academy in 387 BC (Nuwer 1999). Since then, public attitudes around hazing have shifted and currently forty-four states have laws on the books to criminalize and deter hazing (StopHazing 2022). Hazing, much like information, is a term with many definitions. Most contain three elements: a power in-balance, abuse, and initiation.

> Hazing is the exercise of control or power, usually in a punitive or abusive manner, by persons who are members of a group or organization and directed at individuals seeking admission to the group. Hazing often is an initiation ritual. (Guynn and Aquila 2004, 1).

There are also distinctions in the literature between mental hazing, physical hazing (Nuwer 1999), sexual hazing (Guynn and Aquila 2004), and academic hazing (Costa Pinto et al. 2020). Academic hazing is the most similar to information as it is within the context of initiation and transition into higher education.

A few notable attributes of hazing in general, which also apply to information hazing include:

- it can happen regardless of malicious intent (Allan and Madden 2012);
- those who have been hazed believe it is their right or even responsibility to haze others, and that it is used to maintain the values, principles, and exclusivity of the group (Guynn and Aquila 2004).

Along with exclusivity, hazing emphasizes conformity, making mass initiations particularly difficult for members of a potential group who are different, including but not limited to members who do not belong to the dominate race, sexual orientation, gender, appearance, or ideology. Finally, much like the topics throughout this book, hazing is tied to misinformation, malinformation, and deceit.

> Nearly all acts of hazing involve deception. Hazers lie all the time to newcomers. First they lie about the severity of the hazing, which is intended to build fear in the initiate. Then as newcomers invest more and as the initiation process nears an end, hazing escalates, always remaining secret. Hazers also lie to one another, to adults, and to themselves to rationalize that the brutal practices build group unity. (Guynn and Aquila 2004, 26)

Applying this framework onto a critical study of early computing education allows for a more nuanced lens in which to consider the possible impacts of certain policies and procedures that create an unwelcome and exclusionary academic environment for many students.

GOVERNING KNOWLEDGE COMMONS

Governing Knowledge Commons (GKC) is a descriptive framework for analysis, defining knowledge commons, and further investigating how the people within that knowledge commons manage governance. The classroom space has been discussed by others as a knowledge commons (Madison, Frischmann, and Strandburg 2009). Classrooms operate under many agents of control, but the syllabus is one of the most direct and personal factors between the instructor and the student. Instructors often have a wide range of options for requirements and rules with little oversight or review. These factors make syllabi, as governance documents, extremely interesting for analysis.

We see that information hazing can be enacted by a knowledge commons (within the classroom) by means of governance (the syllabus). As a document directly related to control, power, and punishment, the syllabus uses policy and procedures to enact this control at scale. Control over behaviors, specifically around information access and use, aligns with the GKC framework of analysis while still creating a clear link between the framework and our novel concept of information hazing.

Previous works on commons governance have identified the university and its various units and departments as constructed cultural commons where knowledge is produced and disseminated throughout various levels of classrooms, research labs, and administrative level (Madison, Frischmann, and Strandburg 2009). We build on this work and its focus on the formal and informal institutional structures that influence a knowledge commons including policies, structures, and social norms. For our exploration of information hazing in top-tier computer science classrooms, we view the contextual space of higher education, research universities and small liberal arts colleges and universities, and the physical, digital, social, and structural spaces of the

classroom. The classroom and its related digital spaces are our action arena. The actors in our exploration are the students, instructors, and teaching assistants who populate introductory computer science education spaces. The resources that move throughout this commons space include the informational and intellectual material that constitutes the content of the class as well as the information that makes up the policies and rules that constrain the learning environment.

Syllabus as Governance

Outside of direct observation, understanding the power dynamics of any classroom is difficult. Students' lived experiences within these spaces is an essential element of such a description. However, our focus is on examining the governance structure rather than field observation. The course syllabus defines the elements of power, punishment, and control over the classroom. They are presented to students as the governing documents for individual courses. Being used in partnership with any existing student codes, handbooks, or other policies, syllabi have a responsibility to inform students of instructor expectations and policies for that class. Faculty have indicated that their academic integrity policies were sourced from a variety of levels within their institution, often being "defined, redefined, or interpreted at multiple levels" (Simon et al. 2018, 116). The syllabus should then be the final operationalized product of these influences, representing the official rules for their particular space.

Hazing behaviors often appear within the initial onboarding or other introduction to a space. The notion of an introductory or other "first" programming (CS1) course and the resulting classroom (virtual and physical) fit these frameworks as a knowledge commons with formal governance structures, but also serve as a first or early experience within a professional community of programmers The syllabus and classroom connect as "introductory" at two levels: the syllabus as an introduction to the classroom at the micro level and the course itself being the introduction to the field at a macro level.

We examined introductory courses to better understand and describe the interplay between governance and hazing. Our working definition required each course to be a student's initial exposure to programming topics, without any substantial prior programming experiences. These tend to have similar topics across many institutions, allowing us to focus on any differences in governance and information.

A full course load for many undergraduates is between four to six classes running concurrently each semester, meaning that students often operate under four to six different sets of classroom policies that also exist under departmental or program policies. The academic term changeover means that expertise in specific policies is reset as new courses and policies come into effect. Expertise can be difficult to gain, particularly for rarer events like academic misconduct.

Close examination of syllabi within this educational space provides key evidence to observe governance of these knowledge commons, while maintaining our lens of

information hazing on novices. Previous research (Simon et al. 2018) surveyed and interviewed faculty members on their academic integrity policies, which were gathered as part of data collection. These findings provide a baseline to validate our sample and observe differences. They found that faculty used informal explanations alongside official policy: "some noted that the policies are not uniformly enforced or, more commonly, that the policies are customized – officially or not – for a discipline, a course, or even an individual assessment" (Simon et al. 2018, 116). Thus, we can expect that statements included within our samples to be incomplete as compared to what students may be presented within the classroom, but provide a fuller picture of what a faculty member would be required to enforce.

Academic Integrity

Academic dishonesty policies exist to provide definitions and explain consequences, and encompass things like cheating on exams, using fabricated data, forgeries, and plagiarism. These governing documents are created with a variety of names, but students are expected to understand and abide by them. Smaller units, such as departments within a university, often create their own versions meant to work in conjunction with existing policies or provide contextual clarifications. These higher-level policies are broad and meant to cover issues across many courses. The lowest level of these policies are the ones for each course or section and are generally required for each course. As discussed, the syllabus stands as a governing document for the physical classroom space but also the learning space where any work is being completed, including outside the classroom.

The higher-level policies often exist within a system for processing accusations and sanctions. Some academic units may allow faculty members to resolve cases of academic integrity independently, but additional groups exist to process appeals and provide other kinds of oversight. These systems are generally very formal and can have names like the "Honor Council" or "Integrity Committee."

These are formal systems with specific governance charges to uphold. Yet the policies that touch students directly come from the syllabus, where the faculty attempt to remind students of the policies and provide contextual understanding. This project looked at syllabi from introductory computer science courses, so the faculty are expected to provide instruction on how the policies apply to writing computer code and introduce students to (very probably) a new way of handling attribution and citation.

Within Computer Science

Computing work generates a variety of written content, such as reports, documentation, presentations, and books. These items exist to support and explain the other items: source code, executable content, data, and more. Most domains have content needing some special handling, but the written content related to computing works

well with traditional plagiarism, attribution, and crediting paradigms. Source code and other outputs challenge the traditional approaches of detection. New definitions are needed because, unlike with prose writing, novelty and originality appear differently within source code (Simon et al. 2013). Issues related to data (CODATA-ICSTI Task Group on Data Citation Standards and Practices 2013) and source code (Smith, Katz, and Niemeyer 2016) citation are current areas of research, particularly for those working in scientific reproducibility (Vicente-Saez and Martinez-Fuentes 2018). Source code citation and attribution within introductory computer science homework shares some of these similar concerns but is much more focused on issues of plagiarism and detection rather than reproducibility or citation style and object identifiers.

Existing policies around academic integrity for assessments that involve writing prose are insufficient to cover how much of this new content is being used. Altering a quotation from literature may make it no longer blatant plagiarism, the same is not the case for code. Content copied directly from another source almost always requires at least some minimal adaptation to slot into an existing program, even if your intent is to keep the core behavior the same (Wu et al. 2019). Providing attribution for reused code is considered important within computing, but methods for providing such attribution are highly dependent on context and commonly accepted standards (Simon et al. 2016), making the practice not obvious and not represented under existing general plagiarism guidance beyond simply asking students to provide attribution.

COLLECTION METHODS

Sample Design

Our goal was to review a broad collection of introductory computer science syllabi, hoping to observe many forms of information governance and document standard structural patterns.

We restricted our sample to United States higher education institutions and focused on two groups: "Top" ranked undergraduate computer science programs according to US News and World report (Table 4.2) and top programs within small liberal arts colleges (SLAC) (Table 4.1). The US News and World report rankings for 2022 were obtained and saved for later reference. The actual ranked value of each institution was not retained or considered during this investigation. These labels are not mutually exclusive and not intended to be used as such.

SLACs tend to lack representation within nationally ranked lists of colleges by major. Lists of top SLACs for computer science (CS) were obtained from four separate sources.[1] Institutions listed within these lists were combined, and the number of times each appeared was tallied. The final list of candidates was obtained

[1] www.collegetransitions.com/dataverse/best-small-colleges-for-computer-science. www.collegevine.com/schools/best-liberal-arts-colleges-for-computer-science. www.koppelmangroup.com/blog/2019/11/4/the-best-small-liberal-arts-colleges-for-computer-science. https://academicinfluence.com/rankings/discipline/best-computer-science-liberal-arts-colleges.

TABLE 4.1 *List of courses from SLACs in sample*

Institution	Course	Term	Title
Amherst College	CSSC 111	Spring 2020	Intro to computer science I
Bowdoin College	CSCI 1101	Fall 2019	Intro to computer science
Bowdoin College	CSCI 1103	Fall 2021	Programming with data
Carleton College	CS 111	Winter 2019	Intro to computer science
Colgate College	COSC 101	Spring 2022	Intro to computing I
Grinnell College	CS 151	Spring 2022	Functional problem solving
Harvey Mudd College	CS 5	Spring 2022	Intro to computer science
Mount Holyoke College	CS 100	Fall 2019	Intro to computer science
Mount Holyoke College	CS 151	Spring 2022	Introduction to computational problem solving
Swarthmore College	CS 21	Spring 2022	Intro to computer science
Vassar College	CMPU 101	Spring 2020	Problem solving and abstraction
Wellesley College	CS 111	Spring 2022	Computer programming and problem solving
Wellesley College	CS 115	Spring 2020	Computing for the sociotechno web
Wesleyan University	COMP 112	Fall 2017	Intro to programming
Williams College	CSCI 134	Spring 2022	Intro to computer science

TABLE 4.2 *List of courses from top-ranked computer science programs in sample*

Institution	Course	Term	Title
Carnegie Mellon Univ	15–112	Spring 2022	Fundamentals of programming
Cornell Univ	CS 1110	Spring 2022	Intro to computing using python
Cornell Univ	CS 1112	Spring 2022	Intro to Ccmputing using MATLAB
Georgia Institute of Tech	CS 1301	Fall 2017	Intro to computing
Princeton Univ	COS 126	Spring 2022	Computer science: an interdisciplinary approach
Stanford Univ	CS 101	Spring 2022	Intro to computing principles
Stanford Univ	CS 106A	Spring 2021	Programming methodologies
Univ California Berkeley	CS10	Spring 2022	The beauty and joy of computing
Univ California Berkeley	CS61A	Spring 2022	Structure and interpretation of computer programs
Univ of Illinois	CS 105	Spring 2022	Intro computing: nontech
Univ of Illinois	CS 101	Spring 2022	Intro to programming for engineers and scientists
Univ of Michigan	EECS 183	Fall 2020	Elem programming concepts
Univ of Texas Austin	CS 312	Fall 2021	Intro to programming
Univ of Texas Austin	CS 303e	Spring 2022	Elements of computers and programming
Univ of Washington	CSE 142	Winter 2022	Intro to computer programming I

by selecting all the institutions that appeared on at least two of these lists and were not present within the US News and World report ranking group.

The use of "highest ranked" programs within our search boundary has multiple reasons. Top programs across many types of institutions should provide an opportunity to find examples of creative nontraditional practices and possibly faculty who are engaging in currently understood best practices within the computing education field. More practically, we believed these programs had the highest likelihood to have syllabi readily available within their general online resources. We believe that these highly respected programs would serve as an informal standard that other institutions would look toward. With openly available syllabi, we not only gain access but other programs looking to adopt similar practices would as well.

Syllabus Collection

Computer science course descriptions were reviewed for each institution. Specific courses were selected according to the following rules:

- must be a general programming instruction course (versus general computer skills, business applications, etc.);
- may not be a hybrid course covering programming and statistics (e.g., data science courses);
- presume no prior personal or academic programming experience;
- not have any specialty hybrid content denoted in the title;
- multiple courses were allowed so long as the target audiences were fundamentally different.

Tables 4.1 and 4.2 describe the identified courses from these institutions.

Data Analysis

Once all the documents were collected, we used Atlas.ti qualitative coding software to analyze the documents. We applied thematic analysis (Braun and Clarke 2006) to identify policy language, descriptions of cheating, collaboration, and other attributes related to out-of-class learning. Thematic analysis provides a flexible framework for identifying, analyzing, and reporting themes in qualitative data, including documents. Thematic analysis, unlike many other qualitative analysis frameworks, is not tied to any larger theory or narrowly drawn process. We went through the six phases of thematic analysis beginning with familiarizing ourselves with the data, generating codes together through discussion based on the documents and our past familiarity with research literature and experience as instructors in higher education. We then considered the larger trends and themes in the codes, discussed them and their relationships to each other and the research literature. While we focused on open coding, we also found connections with our codes and the GKC, further supporting

our findings that the classroom does represent a knowledge commons space and syllabi are useful artifacts to observe many of these structures. Finally, producing this chapter to where we hope to provide illustrative examples that highlight the opportunities and concerns, we see in this relevant cross-section the best of computing education in higher education across the US.

FINDINGS AND DISCUSSION

Policy presentation within the courses was quite varied, with many different section headers, placements, and so on. Most were introduced using a mix of labels like academic integrity, plagiarism policy, honor code, and collaboration policy. Some linked to separate pages dedicated to specific policies or longer documentation on their expectations for allowable group work, and so on. Many linked to, referenced, or directly quoted the honor code or student code from their department or institution. Most of these went to campus-level honor codes or similar policies, meant for general academic work, but a few had computer science or programming specific guidance.

Observed themes include:

- using behavioral guidance rather than definitions;
- framing peers as information sources to be feared;
- white, western ethics as the only moral compass.

Our findings highlight many of the strategies instructors take to explain these policies, including the variance of approaches and (natural) stumbling points of attempting to work through such a difficult task. Our lens remains focused on understanding the student impact that goes beyond simply understanding the policies and acceptable behaviors.

Advice, not Governance

Plagiarism policies within syllabi serve to provide instruction to students about producing original work and respecting the work of others. These normally remind students to focus on creating original content, use quotations to reference another author's words directly, paraphrase as needed, and so on. Defining these concepts is difficult and an ongoing area of discussion among instructors and researchers as new tools for content creation are adopted. Simon et al. (2018) investigated the language within plagiarism policies for computing classes and stressed that instructors should not depend on student understanding of these concepts even for prose writing, and that there is a strong need for explicit instruction and contextualization of these policies for code writing work.

Do Your Own ~~Work~~ Typing

Terms related to ownership were commonly used within these plagiarism policies, matching many traditional definitions of prose originality. A writer has their "own

voice" and writes in their "own words," for example. Attribution, crediting, quotations, and other techniques are used to respect this ownership when an author's words are reused or for a work reference. Plagiarism can be considered theft of these ideas or fraudulently presenting another person's effort as your own. Most began their policies with some instruction to only submit assignments containing their own original content.

> Assume all work you hand in is to be yours and yours alone. (Wellseley CS 111)
>
> All the work that you submit must be yours. (Mt. Holyoke CS 100)
>
> Work you submit must be predominantly your own. (University of Washington, CSE 142)

Other policies focused on the creative actions taken to create the work, implying that the act itself generated the originality and ownership. These generally add some clarity by specifying the expected actions that the students should be doing alone, but some still fall back to referencing ownership.

> Each individual programming assignment must be coded by you. (Georgia Tech CS 1301)
>
> You must write your own solutions. (UC Berkeley 61a)
>
> The solutions you write up and submit must be your own. (UT Austin CS 312)
>
> Claiming or submitting as your own any work or code you did not fully author, explicitly or implicitly, no matter how small. (CMU 15–112)

The last example (CMU 15–112) is from a long list of actions that are considered academic integrity violations. Not only is "fully author" not defined, but "explicitly or implicitly" were confusing modifiers. Students may interpret this to say that each line of code they produce must be completely unique and original, no matter how small. Many core programming patterns contain very short lines of code that have little to no room for alterations.

Interestingly, many used the action of typing the content into the computer as a replacement for the more creative actions. This is an objective measure and provides an actionable direction to follow.

> You should type and debug all the code you submit. (Stanford CS 101)
>
> You have to be the one that types all of the work that you do in this class . . . You should be the one entering responses to the textbook, pre-lecture activities, and homework. (UIUC CS 105)

Behavioral guidance was commonly provided in lieu of a specific definition for original work. Originality is difficult to explain and gauge, even in the worlds of art, music, and writing, for example. This is compounded with the challenge of explaining it to undergraduate students who likely still struggle with fully understanding

plagiarism requirements for prose writing. Formal definitions may not be easily understood by the students or explained by instructors. Behavioral examples may be a fall back for faculty struggling to provide contextual precision from general descriptions provided by their institutions but represent a very realistic workaround the issues within policy language.

The use of typing as a litmus test for original work was an interesting way to provide some objective measure for a deeply complex and subjective concept. This direction is relatively simple to explain to students and based on an everyday action that they would understand. The implicit meaning behind this rule accurately acknowledges that influence and inspiration for writing the code likely came from many sources. So long as the work was typed "by your own hand" it will be deemed acceptable.

Even when using specific examples, some held brief descriptions with little extra clarity. Verbal guidance might be given informally during class time, but the syllabus remains the primary governing document and is usually treated as the final word when disagreements occur. Similar short policies were found by (Simon et al. 2018) and were described as "snippets of advice" (118).

When documented governance rules are unclear, those that prohibit a strong knowledge commons from forming prevent essential knowledge sharing among peers. Students may attempt to read between the lines and fill in gaps with representations of their own risk tolerance, policies from prior courses, or their knowledge of common industry practices. Requiring students to use their own sense of risk when interpreting the meaning of a policy will likely result in very different interpretations. Students from over-policed groups will be viewing academic policies through the lens of their lived experiences, likely resulting in very different interpretations than peers from under-policed groups.

Biased Threads

Faculty members usually have a wide range of freedom to contextualize policies and requirements as needed for their courses, while also utilizing the course management tools purchased by the wider university. Changes may stem from using different types of assignments or teaching techniques but may also be based on personal preference. Policies on late penalties, for example, can be a large area of difference. Some faculty see being late to class or late submitting assignments as a form of professional disrespect while others see one or both as a less of a matter to be punished, and more a byproduct of life. These differences are often due to personal experiences or professional expectations within their discipline. Personal preferences like these often end up encoded within policies.

These differing preferences among faculty are seen across the syllabi in our sample, even within the same institution. Some suggested that solutions and related code were completely forbidden and tainted any original work the student attempted to produce, while another directed students to use the solutions to complete the assignments.

> Looking at solutions from other students or any other source (including the web), or collaborating to write solutions to individual work, is considered a violation of the honor code. (Mt. Holyoke CS 100)

> Homeworks are an opportunity to engage with the material and practice your programming skills. To support you, solutions will be available on moodle. It is your responsibility to engage with them in a way that strategically moves your learning forward. (Mt. Holyoke CS 151)

These students will have dramatically different experiences and expectations for future programming courses. One student may leave feeling like they missed out on the important reflective nature of working with open solutions, while another is not used to the isolation and struggle of closed material problem solving.

Faculty customizing policy choices allows them the freedom to quickly adapt to students and situations, try things out, and find the right fit for their content. Our own course policies differ based on our preferences and the kind of content that we teach. Differences in belief about collaboration versus independence is a specific pedagogical choice, but other areas of difference may allow for misconceptions or bias to appear in policy.

Source plagiarism detection systems are commonly used to compare submitted code to other submissions and other known solutions. While there are some differences in how each system processes and calculates similarity, the core action is comparing the text patterns within files. These tools appeared to strongly influence the policy guidance given for each course. Many policies mentioned using a detection system and described them as systems to detect cheating or plagiarism. Rarely were they mentioned by name or other details on how violations were measured.

Viewing source code and prose writing as the same type of creative activity is common and generally unproblematic. However, treating these two things as the same when attempting to define attribution and plagiarism creates problems. Plagiarism in source code is nothing new, and has been discussed long before these detection tools existed. As discussed by Gibson (2009), providing attribution for reused or referenced source code requires adaptation depending on context. He named his guidance a "code of practice" because he wanted it to be seen as a style guide where source code authors could make informed decisions rather than a strict set of rules. When used in a normal context of a software project or company, programmers have the freedom to make these contextual choices.

Some policies directed students to existing academic writing policies for their programming assignments, even directly comparing writing code to writing essays. These materials remain valuable for prose writing but usually lack guidance for source code attribution if they even discuss source code at all.

> Don't cheat. Read *Originality and Attribution: A guide for student writers at Vassar College*. Copying someone else's code is plagiarism. School policy dictates

> instructors must report all suspected incidents of cheating to their department chair. Please don't put yourself or your professor in that position. When in doubt, ask your professor before seeking help from another source. (Vassar CS 101)

> One of the key things to understand about programming, and computer science in general, is that it is a writing-heavy discipline. When you create a computer program, you are writing a document, just like you write documents in an English class (or any class that involves essays). Therefore, many of the same rules that apply to writing essays also apply to computer programs, particularly regarding plagiarism. (Cornell CS 1110)

This approach appears to be an honest but common misconception that goes directly against known and documented limitations of this approach. Given their lack of awareness, experience, and guidance, novice programmers could easily violate these policies without knowing it (Simon et al. 2014). Riedesel et al. (2012) stressed the need for institutions to have a well-understood policy for source code plagiarism, while Simon et al. (2018) found that instructors will still amend it with their own preferences and contextual needs. Discussion about this issue has grown recently, making it unlikely that a faculty member would discover it otherwise unless specifically searching for it.

The Vassar (CS 101) example below is notable because this quotation is the entirety of the academic integrity policy section. Additionally, the book mentioned appears to be from 1980 and only accessible from an administrator's office. Students are told to read a writing handbook for guidance about source code attribution, when the handbook contains no reference to programming, so they end up with no actual guidance. Additionally, these students may believe that it should be there, but they lack the skill to find it.

> Plagiarism is relatively easy to detect in a programming class. Do not take shortcuts. Always do your own work. Note that we encourage discussion on course content. (UIUC CS 101)

Some policies gave direct instructions on how to adjust their study behaviors and conversations to better fit with how these systems work. Code or notes resulting from student discussions were often restricted in addition to conversations. Many explained that written notes allowed code from each student to look too similar and could cause it to be flagged as plagiarism by detection tools.

> Students sometimes unwittingly get "carried away" and start writing code discussing a solution to an assignment, only to turn in suspiciously similar code that is flagged by our plagiarism detection system, which is then subject to academic dishonesty policies. (UT Austin CS 312)

One policy even went as far as suggesting that students wait thirty minutes after any interaction to write any code as a method to prevent accidental replication between parties.

> A good rule of thumb to ensuring your collaboration is allowed is not take written notes, photographs, or other records during your discussion and wait at least 30 minutes after completing the discussion before returning to your own work. (You could use this time to relax, watch TV, listen to a podcast, or do work for another class.) For most students, this will result in you only bringing the high-level concepts of the collaboration back to your work, and ensuring that you reconstruct the ideas on your own. (U Washington ESE 142)

Remembering elements of code from conversations or other indirect connections were not covered within working definitions of source code plagiarism (see Novak, Joy, and Kermek 2019). Yet producing solutions that were too similar due to this was explicitly labeled as cheating because the detection system flags these similarities. The detection systems are literally defining the forms of cheating within these courses.

Policies based on misconceptions can be more than just misinformation. Directions instruct students seeking to follow the rules that help exist within certain documents. Finding no actual reference to their problem has several consequences. In the short term, the student may not meet expectations due to lack of guidance and question their own understanding or seeking out help again. Longer term, the frustration of being directed to use nonexistent information may deter the student from progressing with a subject area.

While the comparison between source code and prose may be simple misinformation, telling students to seek help within policies containing no guidance on source code attribution is malinformation at best and gaslighting at worst.

Certain Uncertainty

Concerns about communication and access to prohibited information were common. The previous section covered the actions faculty required and suggested students perform to ensure originality. Discussions of communication came up in elements related to the plagiarism detection systems, but this next section looks deeper into issues of communication. Student conversations were strongly policed by many policies and faculty used policy space to add their own moral judgments and threats about cheating and plagiarism.

Code (Of/In) Silence

Faculty used words like "collaboration," "conversations," "getting help," and "discussions" within policies to describe interpersonal interaction between students about the homework assignments. These descriptions took up most of the policies and provided rules for interactions: specifying the modality, content, and even language. Student collaboration to complete a group assignment was usually differentiated from more generally talking about the assignment content, with the informal conversations being the instructor's largest concern.

This policy area had some of the broadest set of rules and requirements for students, ranging from effectively no restrictions to only speaking to instructional staff for the entire term. These policy sections had the most examples and detailed descriptions of expectations. Most policies allowed or encouraged some form of discussion between students and gave details on the range of acceptable topics for their interactions.

> Note that we encourage discussion on course content. (UIUC CS 101)

> Discussing ideas and approaches to problems with others on a general level is encouraged. (Swarthmore CS 21)

> We believe that collaboration fosters a healthy and enjoyable educational environment. For this reason, we encourage you to talk with other students about the course material and to form study groups. (Wellesley CS 115)

Programming assignments can involve substantial problem solving, research, trial and error, and resolving bugs within the code. Students struggle with these areas throughout CS1 courses, making them common conversation topics within a classroom and learning group. Specific areas like approaches for problem solving, programming concepts and patterns, and seeking help from peers were usually broken into clusters approved for considerations or any off-limits topics.

> In most cases, you may discuss concepts (algorithms, ideas, approaches, etc.) described in the readings, lab exercises, or during class with anyone. (Carleton CS 111)

> You are free to discuss ideas and approaches with other students, and then implement the solution yourself. (Stanford CS 101)

> We recognize that collaboration can help you master course material. In fact, there are certain ways in which we will encourage you to collaborate. These include: discussing course content at a high level, getting hints or debugging help, talking about problem-solving strategies, discussing ideas together. (Mt. Holyoke CS 151)

The modality was usually required to be in person with verbal communication. The language descriptions often had the modifiers of "natural" or "human" language as a method of restricting anything like a programming language to be spoken aloud.

> Such a description should be in English or another natural human language. (UW CSE 142)

> Verbal collaboration without code sharing . . . (Bowdoin CS Collaboration Policy)

> You may collaborate with other students by talking about the problem or your solution in a natural language (e.g., English), but you may not use any formal language, and especially not Python. (Wellseley CS 111)

> Roughly speaking, it is okay to share ideas but it is not okay to share any artifacts (code, write-up, etc.). Here is a good way to think about it: you and a classmate can

> get together, discuss ideas, and even write some code. However, you are expected to leave that meeting with nothing – no notes and certainly no code – and write up your own solution. (Colgate COSC 101)

This final example (Colgate COSC 101) gives some insight into why there were these requirements. Code is plain text, making it very easy to share within text platforms. Students often have group chats or another text-based communication setup, so sharing portions of their code is quite easy and could possibly include a dozen students as recipients. Limiting the language spoken generally forces students to speak in broader, more conceptual terms, rather than specific elements of the assignment being discussed. These two specific behavioral requirements encompass many common concerns without lengthy behavioral lists.

Content and modality were not the only limited areas for interaction. Specific groups of people were also prohibited. A few mandated students to hold no homework-related interactions with anyone other than approved course staff. While very strict, this rule was directly stated and essentially eliminates collusion.

> Discussing any part of assignments or assessments with anyone else (besides 112 TA's and course faculty), no matter how briefly or casually, in person or via Discord/WeChat/Zoom etc. unless the assignment explicitly allows it … all homework exercises are solo, meaning that you must not collaborate in any way. (CMU 15–112)

> Students may only collaborate with fellow students currently taking CS 1301, the TA's and the instructor. (Georgia Tech CS 1301)

Peers were commonly presented as a source of seeing, hearing, or receiving forbidden information and collaboration turning into collusion. These two concepts are intrinsically related behaviors, as they are referring to the same set of social actions that people may take toward a project (Fraser 2014). Collaboration describes all the possible social actions taken toward a project or goal, while collusion covers any of those actions that are not allowed. This means that the very definition of collusion depends on the rules and scope for collaboration to be defined.

> There is a difference between learning collaboratively and completing the work for someone else. This can be a subtle but important distinction. … The following types of collaboration are encouraged … The following types of collaboration are prohibited and may constitute academic misconduct. (UWashington CSE 142)

> You may not share information about your solution in such a manner that a student could reconstruct your solution in a meaningful way (such as by dictation, providing a detailed outline, or discussing specific aspects of the solution). (Swarthmore CS 21)

> All assignments must be done individually. No collaboration of any kind is allowed. Any cases of inappropriate collaboration (cheating) have to be reported to the department chair and the Academic Panel, and will be dealt with promptly. (Vassar CS 101)

Many attempted to define the level of detail students could include in their conversations, hoping to highlight the tipping point into collusion. While not formal definitions, providing examples of unacceptable behaviors is an appropriate way to approach this challenge. Unfortunately, many examples explicitly told students they should already know what is meant by these labels.

> Collaboration and learning from one another are encouraged. Copying and cheating are strictly forbidden. You are expected to be able to distinguish the two. If you're contemplating an action and you're not sure into which category it falls, you should consider whether what you intend to submit is the product of your own efforts and represents your own understanding of the ideas involved. If it is/does not, then you should not submit the work as your own. Wesleyan imposes an Honor Code (to be found in the Student Handbook). You are expected to abide by it in all of your courses, including this one. (Wesleyan COMP 112)

Encoding very fine lines of communication into clear and actionable definitions is not an easy task. Varied interpretations (Joy et al. 2013; Sheard et al. 2002; Simon et al. 2013) and operating definitions (Fraser 2014; Riedesel et al. 2012) about academic integrity exist within computing faculty and students. Faculty are often left solely responsible for writing contextualized governance. Lack of specificity in policies passed to the faculty seems to be a related issue, where faculty "find their role to be one of contextualization" (Simon et al. 2018) of the policies they are given. Several of their respondents mentioned using behavioral examples because policy language alone was insufficient for students to understand expectations, which was also found within our sample. Providing these examples opened other areas for misinterpretation.

> Looking at code related to an assignment together is considered crossing the line into cheating territory! (UT Austin CS 312)

This example (UT Austin CS 312) came from the middle of a large paragraph (about 430 words) that represents just a portion of the academic integrity policy content of the syllabus. The paragraph gives numerous examples of cheating behavior, including the example above. The phrasing "code related to an assignment" is likely intended to mean looking at another student's code for an assignment. This opened a reasonably descriptive policy to potentially include any class sources related to the assignment. Students carefully reading the policy and taking this statement very literally could naturally include things like the professor's notes, textbook, or even their class notes. Each of those elements likely contain elements for the homework.

Peers were often presented as a source of forbidden knowledge and just asking for help could potentially result in harm to both parties. This harm ranged from formal sanctions due to violating policies to informal where learning opportunities were missed.

> Please don't put your friends at risk by asking them for unauthorized help. (Cornell CS 1110)

> You rob yourself and others of learning how to approach difficult programming problems, an essential skill for future classes. (UC Berkeley 10a)

> Providing help beyond what is allowed is as much of an infraction of the honor code as receiving help. (Bowdoin CS Collaboration Policy)

As several syllabi reported, once a student sees an example of the solution, their memory and ability to perform "original" work has been forever tainted. Students must complete the problems without seeing external sources and are instructed within the policies to maintain the isolation, protect their work, and to avoid situations where they may inadvertently expose themselves or others. While some policies allow for students to cite what they have looked for or other sources of help, some disallow nearly all forms of communication. These environments may differ between instructors and institutions such that not all grades and assessment outcomes may be considered equal. While one student has the benefit of a study group or partner to discuss concepts and concerns with, another may only access help during office hours or email with course staff.

Students were also warned to protect their work from peers who may maliciously view and copy content. An unknowing victim would receive the same severity of consequences as the student who took the direct action to copy the work.

> You are expected to take reasonable measures to protect your work from unauthorized access by others, including your electronic files, print-outs, and written work. (UT Austin CS 312)

This presentation of peers as being sources of constant potential risk isolates students and creates an oppositional existence instead of collegial. However, this may only be the case for certain groups of newcomers. Students with higher risk tolerance and a better understanding of higher education practices may not feel the same way about their peers and are able to more freely engage in group studying and social support for learning. Their lowered risk awareness also makes them a riskier person to socialize with for students afraid of forbidden information exposure, further expanding their isolation.

With collaboration and originality presented as mutually exclusive, compounded with the individual academic assessment justification, peers tended to be described as objects or sources to be feared. They are potential sources of exposure to external ideas and code, but also active threats of observation. Several policies instruct students to actively maintain a two-way barrier from viewing and being viewed. This includes incidents where another student deliberately uses someone else's laptop to see the answer. The laptop's owner is deemed just as guilty for not locking down their computer.

Normal plagiarism policies for academic writing refer to overt actions taken by students: the inclusion of text from another work without proper attribution, purposefully copying text, and others. A purposeful choice, even if not understood to be

an infraction, can be pointed to as the problem. Treating passive infractions as overt and deliberate choices to cheat requires students to be constantly diligent of the actions and intents of their peers. The result may look the same: Code submitted by students could be tagged as possible plagiarism by a detection tool, but the reason is left unclear without investigation. Students and faculty end up navigating this confusing combination of understanding the tool, the policies, and behaviors.

Importing Morality

The origin story of any syllabus involves reuse, boilerplate requirements, and educated guesswork. Faculty may inherit policies from previous instructors of a course, use a template with a campus or department boilerplate, and then add their own content and personalization. Once created, review services may not be offered or required before the syllabus is used for a class. Errors or problematic language easily fall through the cracks.

Although not generally required, many faculty elected to include justification and reasoning behind their policies. These sections had some of the most informal and emotionally changed content out of all the content reviewed for this project. Many intended to defend the policy choices, explained why it was worth following, and highlighted the severity of the sanctions. Some went further and presented moral values or other ethical incentives for abiding by their governing rules. These justifications generally fell into two categories: ethics and respect or learning goals.

Justification claims under ethics invoked a variety of concepts in their persuasive arguments: honor, respect, fairness, honesty, personal responsibility, ethics, moral codes, and so on. The objects these concepts acted on were often a larger social group, such as the institution, learning community, and peers. Positive policy examples set a tone of pride for students to adopt and maintain.

> For your sake and the sake of the Claremont-Colleges community, please conduct yourself with the highest level of academic integrity. (Harvey Mudd CS 5)

> Simply put, academic integrity is about respecting yourself and respecting others. You respect yourself by submitting work completed through your own effort; you respect others by acknowledging contributions from others when such external contribution is allowed. (Cornell CS 1112)

The implication of these arguments was left unsaid: that violating this respect can cause harm to yourself; but it also threatens to withhold trust or approval of the students should they break these rules. Negative examples focused more on this consequence.

> Sadly, we encounter such violations almost every semester, and the rulings of the Honor Code Council tend to be harsh. For your sake and ours, please don't be tempted to cheat; you are likely to be much better off getting a poor grade on an problem set or exam than you are if you are found guilty of cheating! (Wellseley CS 111)

> We begin by choosing to trust each of you individually. Do not be one of the few who loses that trust. If you cheat, expect to be caught, and expect significant consequences. Use common sense and understand these rules. (CMU 15–112)

These may lack overt emotionally charged words, but describing these acts as being forbidden, learning shortcuts, and plain cheating set a judgmental and threatening tone. Many of the longer policies contained emotional discussions of the morality around cheating, yet the amount of actionable information about cheating was quite low.

> Many students begin every assignment by immediately going to Google, trying to find something that might keep them from having to solve the problem for themselves. *This is an incredibly stupid thing to do* . . . you're starting down a moral slippery slope that's liable to send you over a cliff . . . a complete solution that some idiot has posted on GitHub; it will be too tempting not to use it . . . *Don't do it!* If you are caught, you will deeply regret it. And even if you're not caught, you're still a cheating low-life. (UT Austin CS 303E)

> It's not that people can be divided into cheaters and noncheaters in some preordained way, though that is an easy way to think about life. It's more than the stress and bad decision making of a particular *situation* make a cheater of someone. (Stanford CS 106a)

Others maintained a more pragmatic framing around fairness and legitimacy. These focused on the learning assessment process and the importance that grades accurately represent their understanding of the content. Cheating or subverting the expected values of the policy would make it unfair for the other students following the morally correct path or unfair to your own learning.

> . . . does not contribute toward developing mastery. In addition, this deprives you of the ability to receive feedback and support from the course staff in addressing the areas in which you are struggling. (U Washington CSE 142)

> Cheating not only robs you of an opportunity to learn, it also devalues your peers' hard work. Because of this, we take cheating very seriously in this course. (UIUC CS 101)

Some added computing industry practices to frame the purpose of their requirements. These references were uncommon, but almost all did so to explain why they promoted collaboration.

> We believe that collaboration fosters a healthy and enjoyable educational environment. For this reason, we encourage you to talk with other students about the course material and to form study groups. Programming assignments in this course can be challenging. Also teamwork is the norm in the CS industry. Given the above, some of the assignment work is required to be done with a partner, while some is required to be done individually. (Wellesley CS 115)

> Note: details differ from place to place, but the shared-software-development approach described above is very common in professional practice. Sharing the

> building and understanding of software is the characteristic computing skill of our era! (Harvey Mudd CS5)

Authentic assessment is the concept of a course using assessments that closely match authentic actions within the practice. As discussed by Simon and Sheard (2015), these are valuable for learning but often require higher levels of interaction with the instruction team and are harder to grade at scale and involve collaboration. These factors put adoption of authentic assessment at odds with the focus on individual learning and assessment needed for awarding individual grades.

These justifications also highlight the two general approaches taken toward academic integrity: consequences and virtuousness. The former seeks to leverage punitive measures to ensure that students do not act on the desire to cheat, while the latter attempts to fill in educational and mindset gaps to prevent the desire in the first place.

Introductory students within these restrictive and highly individualistic classes get an inauthentic and overly policed experience within computer science. At a micro level, individual students work under a punitive and unclear policy. At a meso level, looking at the collection of peers operating under these policies, experienced students are put at the advantage both in permissible communication and deepening peer connections. Students with novice level experience work in isolation while being instructed to fear betrayal by peers.

In considering the macro level in all these policies, many students are being prepared to operate in an environment that mischaracterizes the essential operations within this profession. Code is commonly discussed, shared, borrowed, and inherited within teams. The errors and omissions described here may seem like honest mistakes from an instructor navigating a vague space, but we argue that it is precisely this imprecision that is the point. We argue that the hazing element comes from how the policies describe information that should be accessed or restricted. These policies may be unintentionally problematic at an individual level, but when situated among their peers, they replicate those in power who seek to promote same or similar group learners to achieve success and value placing additional checks against marginalized groups.

Encoding White Spaces

Academic integrity does not have a single universal definition. Global and historical traditions intersect as well as discipline specific interpretations. As discussed by (Macfarlane, Zhang, and Pun 2014), discussions and literature in this area can include academic integrity, honor codes, and professional ethics. Synonyms abound and most works spend a nontrivial amount of time clarifying their viewpoint and scope for their research. Growing education globalization, information access, and technologies for text similarity detection have reinforced the need for institutions to

have a shared understanding of academic integrity issues and active training on the topics.

Modern tools to detect plagiarism and other academic dishonesty issues may be presented as objective measures, but each require review and interpretation. This means that the gap between being flagged and being accused is once again expected to be filled by "gut" feelings. Faculty must understand these tools but also navigate their own policies to determine if further action is needed. Vague and open-ended policy description leave decision making susceptible to unconscious biases and subjective opinions. These tools presented as objective measures remain dependent on evaluators to make objective judgments.

Unconscious bias and suspicion put black students at higher risk of being reported by peers for perceived infractions. This is directly discussed by participants in Feagin and Sikes (1995). Participants reported being accused of plagiarism by instructors solely because they did not often speak up in class and the professor simply did not believe their work was original. Those with the power to assess student work may easily be swayed by unconscious biases about the innate abilities of students of color.

Historically white colonial traditions can be found within modern expectations of academic behavior. Bonilla-Silva and Peoples (2022) outlined these acts and their result of perpetuating erasure of nonwhite traditions. Higher education curricula not only stress white content, but also values and traditions. While not directly discussed by Bonilla-Silva and Peoples, the elements of whiteness held up within these institutions can be extended to the policies of the classroom and therefore the syllabus.

The long history of nonwhite students being subjected to the varied feelings and perceptions of faculty members continues to this day and certainly within computer science classrooms. Policies being written and enforced with technical review tools to support them provide a sense of authority and likely make faculty feel comfortable that they are working within a "color-blind" system, but do very little to provide safeguards for students who have historically been subjected to higher levels of scrutiny and mistrust. Without clear process documentation and clear policies to be accused of violating, the power here rests solely within the accusing faculty's perception of the truth and internal definitions of the rules.

Disproportionate impact cannot be measured as a byproduct of the intent of the faculty member writing the policy. Students will act according to the policies as they are given and interpreted through their understanding of rights and risk. This means that each student will have a unique interpretation and thus experience different impacts on their behavior. This alone will always be the expectation, but becomes problematic when certain groups of students will be systematically impacted in different ways along the lines of race, socioeconomic status, and other social factors.

New tools and techniques used in response to these challenges of student growth are constantly created and adopted. Some may have existed a decade ago but were not used at the scale they are at the time of this writing. Creating the code database

mentioned in the example below was not truly possible until recently. Faculty writing these syllabi likely learned how to program under very different systems; most importantly, their work was less likely to be monitored at such a degree.

> Plagiarism detection. We use sophisticated software tools to detect plagiarism. For example, we compare each submission (including interim submissions) against a code database, which includes all submissions from this offering and previous offerings of the course. While we take no pleasure in bringing cases to the Committee on Discipline; it is our duty to keep the playing field level for the overwhelming majority of students who work very hard in this course and follow the rules. (Princeton COS 126)

While these policies are attempting to dismantle a knowledge commons forming within the classroom, it is actually dismantling it only for these risk-averse students. Students with in-group connections can be told or mentored through how to subvert or avoid detection, preserving the knowledge commons just for them. Out-group students must remain isolated and fearful, providing evidence to the in-group members that any outsiders who were able to complete courses like this must be exceptional and worthy of academic respect.

> Utilizing discussion with peers to facilitate your learning is a critical skill for success in computer science. However, at the same time, you must be aware that getting stuck and pushing through challenging problems is essential for robust learning. (Grinnel CSC 151)

The struggle through these introductory programming courses is often seen and framed as a rite of passage, one that the instructor survived and thus so must they. Their survival is framed as tribute to the industry. Yet the struggle and isolation created under some of these policies is a lie. Those working in industry borrow and build on external code constantly. They also work together in teams. While pair programming is a common tool for code review and mentorship, large projects are split up among members of a team who will work on independent elements, but each with the support and guidance of other team members.

Conclusions

Much of the research has been on what people consider misconduct, and whether students committing misconduct are aware they are doing so. We seek to flip this question, and ask: How far might a student go to avoid misconduct out of a fear of committing an infraction? Are the forms of these policies doing a disservice for students in fully recognizing their own behavior as misconduct; but do the policies also open up the possibility for students to incorrectly limit their own behaviors?

Overly restrictive policies also create stressors via isolation and fear. Students are isolated from peers in their work with the content, with the potential to reinforce prior feelings of otherness and being an impostor. As defined by many of the

policies, communication and sharing resources with peers is something to be feared. All of this creates an environment rife for misinformation to spread.

Hazing also continues after the first programming class. These strict policies and foundational fears are cemented during job interviews. Whiteboard coding questions are the epitome of fictional practices treated as a norm within computer science. Presenting handwritten syntactically perfect code on a whiteboard with no tools or resources, under time constraints and observation, creates a high stress environment that represents very little of what the actual day to day job is like. The growing cost of higher education and the student loan crisis make these interviews a high-stakes finale to the overall system of hazing within the computing industry. However, many will complete these efforts because they have been prepared by prior hazing rituals to expect the initiation costs to be high. The other element of the lie that makes this hazing is that all these tasks are meant to be a measure of your worth within a giant meritocracy, when networking, bias, alumni networks, and programming interview cram schools are all leveraged by others to bypass these checks of merit.

Employers are outsourcing the measurement and evaluation of a candidate's skill in the field to the grades assigned during academic evaluation (Simon and Sheard 2015). By outsourcing this evaluation work, they measure elements of hireability as those who are successful within the university grading system. Obviously not the sole factor used by many companies, grade point average (GPA) and other exam ranks do play an important role in determining candidates and finalists for internships and full-time work. Students who have more access to support or more freedom to subvert assessment strategies have a stronger chance of ranking higher and receiving these positions.

Academic dishonesty policies are contributors to creating an environment where students expect to be creating source code in isolated, high pressure, and competitive environments. Many startup companies expect staff to show their dedication to the project by working extreme hours within similarly high-pressure situations. A student graduating out of such a program and landing a role within a company may have little experience or basis to push back on these unhealthy expectations.

Hazing is more than simple gatekeeping, but an effort to ensure that all who pass through come to believe that those filtering structures are valid and important. The goal is not just to reenact them when they later come into their own house of power but to have those in power genuinely believe in those structures and the reasoning behind them. Information hazing, therefore, is setting these structures off and making the filters based on how information is used and accessed by those newcomers. Rather than overt acts upon them or specific demands for action, this hazing is more subtle. It circumvents many of our normal triggers to view something with suspicion or questioning.

The information hazing observed within this study hides behind western ethics and neoliberal morality frameworks. After all, questioning something that is

supposed to uphold honor and integrity must make that questioning be in opposition to those factors. Those not interested in upholding the virtues of honesty must therefore be against it. However, holding the rigid moralistic jargon up to a lens of information use and access allows for objections that are based in tangible ways: that many of these statements are problematic due to wide ambiguity of or a misrepresentation of the natural process of computing work.

In closing, we want to note that although we have highlighted many aspects of information hazing that we believe are likely to cause harm in the classroom and beyond, we do not write with this to blame the instructors who have created, curated, and crafted the syllabi in our sample. We know how hard teaching is and how complicated plagiarism and academic integrity are to teach, enforce, and navigate. We created this work with the goal of identifying problems in order to create positive change, and we hope that this chapter helps us to see the things that we have ignored or thought were the last best option.

REFERENCES

Allan, Elizabeth J., and Mary Madden. 2012. "The Nature and Extent of College Student Hazing." *International Journal of Adolescent Medicine and Health* 24 (1): 83–90. https://doi.org/10.1515/ijamh.2012.012.

Bonilla-Silva, Eduardo, and Crystal E. Peoples. 2022. "Historically White Colleges and Universities: The Unbearable Whiteness of (Most) Colleges and Universities in America." *American Behavioral Scientist* 66 (11): 1490–1504. https://doi.org/10.1177/00027642211066047.

Braun, Virginia, and Victoria Clarke. 2006. "Using Thematic Analysis in Psychology." *Qualitative Research in Psychology* 3 (2): 77–101. https://doi.org/10.1191/1478088706qp063oa.

CODATA-ICSTI Task Group on Data Citation Standards and Practices. 2013. "Out of Cite, Out of Mind: The Current State of Practice, Policy, and Technology for the Citation of Data." *Data Science Journal* 12: CIDCR1–CIDCR7. https://doi.org/10.2481/dsj.OSOM13–043.

Costa Pinto, Ligia M., Carla Sá, Nuno Soares, Sílvia Sousa, and Marieta Valente. 2020. "The Case for Academic Hazing as a Rational Choice: An Economic Approach." *Economic Analysis and Policy* 66 (June): 51–62. https://doi.org/10.1016/j.eap.2020.02.004.

Feagin, Joe R., and Melvin P. Sikes. 1995. "How Black Students Cope with Racism on White Campuses." *Journal of Blacks in Higher Education* 8: 91–97. https://doi.org/10.2307/2963064.

Fraser, Robert. 2014. "Collaboration, Collusion and Plagiarism in Computer Science Coursework." *Informatics in Education* 13 (2): 179–195. https://doi.org/10.15388/infedu.2014.10.

Gibson, J. Paul. 2003. "Software Reuse in Final Year Projects: A Code of Practice." Report NUIM-CS-2003-TR12, Maynooth: National University of Ireland, November.

2009. "Software Reuse and Plagiarism: A Code of Practice." *ACM SIGCSE Bulletin* 41 (3): 55–59. https://doi.org/10.1145/1595496.1562900.

Guynn, Kevin L., and Frank D. Aquila. 2004. *Hazing in High Schools: Causes and Consequences*. Bloomington, IN: Phi Delta Kappa Educational Foundation.

Joy, Mike S., Jane E. Sinclair, Russell Boyatt, Jane Yin-Kim Yau, and Georgina Cosma. 2013. "Student Perspectives on Source-Code Plagiarism." *International Journal for Educational Integrity* 9 (1): 3–19. https://doi.org/10.21913/IJEI.v9i1.844.

Macfarlane, Bruce, Jingjing Zhang, and Annie Pun. 2014. "Academic Integrity: A Review of the Literature." *Studies in Higher Education* 39 (2): 339–358. https://doi.org/10.1080/03075079.2012.709495.

Madison, Michael J., Brett M. Frischmann, and Katherine J. Strandburg. 2009. "The University as Constructed Cultural Commons." *Journal of Law and Policy* 30: 365. https://doi.org/10.31228/osf.io/z8x7m.

Novak, Matija, Mike Joy, and Dragutin Kermek. 2019. "Source-Code Similarity Detection and Detection Tools Used in Academia: A Systematic Review." *ACM Transactions on Computing Education* 19 (3): 1–37. https://doi.org/10.1145/3313290.

Nuwer, Hank. 1999. *Wrongs of Passage: Fraternities, Sororities, Hazing, and Binge Drinking.* Bloomington: Indiana University Press.

Riedesel, Charles P., Alison L. Clear, Gerry W. Cross, Janet M. Hughes, Simon, and Henry M. Walker. 2012. "Academic Integrity Policies in a Computing Education Context." In *Proceedings of the Final Reports on Innovation and Technology in Computer Science Education 2012 Working Groups*, 1–15. ACM, Haifa. https://doi.org/10.1145/2426636.2426638.

Sheard, Judy, Martin Dick, Selby Markham, Ian Macdonald, and Meaghan Walsh. 2002. "Cheating and Plagiarism: Perceptions and Practices of First Year IT Students." In *Proceedings of the 7th Annual Conference on Innovation and Technology in Computer Science Education*, 183–187. ACM, Aarhus. https://doi.org/10.1145/544414.544468.

Simon, and Judy Sheard. 2015. "Academic Integrity and Professional Integrity in Computing Education." In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, 237–141. ITiCSE 2015. Association for Computing Machinery, New York. https://doi.org/10.1145/2729094.2742633.

Sheard, Simon, Beth Cook, Judy Sheard, Angela Carbone, and Chris Johnson. 2013. "Academic Integrity: Differences between Computing Assessments and Essays." In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research*, 23–32. ACM, Koli. https://doi.org/10.1145/2526968.2526971.

2014. "Student Perceptions of the Acceptability of Various Code-Writing Practices." In *Proceedings of the 2014 Conference on Innovation and Technology in Computer Science Education*, 105–110. ITiCSE 2014. Association for Computing Machinery, New York. https://doi.org/10.1145/2591708.2591755.

Simon, Judy Sheard, Michael Morgan, Andrew Petersen, Amber Settle, and Jane Sinclair. 2018. "Informing Students about Academic Integrity in Programming." In *Proceedings of the 20th Australasian Computing Education Conference*, 113–122. ACE 2018. Association for Computing Machinery, New York. https://doi.org/10.1145/3160489.3160502.

Simon, Judy Sheard, Michael Morgan, and Charles Riedesel et al. 2016. "Negotiating the Maze of Academic Integrity in Computing Education." In *Proceedings of the 2016 ITiCSE Working Group Reports*, 57–80. ITiCSE 2016. Association for Computing Machinery, New York. https://doi.org/10.1145/3024906.3024910.

Smith, Arfon M., Daniel S. Katz, and Kyle E. Niemeyer. 2016. "Software Citation Principles." *PeerJ Computer Science* 2 (September): e86. https://doi.org/10.7717/peerj-cs.86.

StopHazing. 2022. "States with Anti-Hazing Laws." Stophazing.Org. 2022. https://stophazing.org/policy/state-laws/.

Vicente-Saez, Ruben, and Clara Martinez-Fuentes. 2018. "Open Science Now: A Systematic Literature Review for an Integrated Definition." *Journal of Business Research* 88 (July): 428–36. https://doi.org/10.1016/j.jbusres.2017.12.043.

Wu, Yuhao, Shaowei Wang, Cor-Paul Bezemer, and Katsuro Inoue. 2019. "How Do Developers Utilize Source Code from Stack Overflow?" *Empirical Software Engineering* 24 (2): 637–673. https://doi.org/10.1007/s10664–018-9634-5.