education sciences

MDPI

Article

Rethinking Undergraduate Computer Science Education: Using the 4Es Heuristic to Center Students in an Introductory Computer Science Course

Francheska D. Starks *, Shalaunda M. Reeves *, Jonathan Rickert, Kyle Li, Brock Couch and Joanna Millunchick

College of Education, Health, and Human Sciences, University of Tennessee, 1122 Volunteer Boulevard, Knoxville, TN 37996, USA
* Correspondence: fstarks1@utk.edu (F.D.S.); sreeve10@utk.edu (S.M.R.)

Abstract: There is a nationwide effort to increase the representation and engagement of minoritized students in computer science education. Discourse about the barriers to diversity among computer science majors is often characterized by student pathologies and does not consider the impacts of classroom culture and instructor pedagogies. Amid the push for strategies to recruit and retain minoritized students in computer science, little has been done to transform curriculum and analyze faculty perspectives on curriculum and pedagogy as methods to increase students' access to the computer science major. This paper presents an example of curriculum redesign for an undergraduate introductory computer science course (ICS) that sought to address issues of inequitable representation by centering student identities and redistributing power in favor of students. The authors draw upon critical sociocultural and the 4Es heuristic for disciplinary literacy to reimagine the ICS course as a space that centers on the important roles of identity and power in solving for diversity in computer science education. We highlight for researchers and practitioners how our work may be used to disrupt meritocratic practices that alienate minoritized and economically disadvantaged students and to expand definitions of mastery and expertise in computer science education.

Keywords: computer science; STEM education; disciplinary literacy; equity; undergraduate education

Citation: Starks, F.D.; Reeves, S.M.; Rickert, J.; Li, K.; Couch, B.; Millunchick, J. Rethinking Undergraduate Computer Science Education: Using the 4Es Heuristic to Center Students in an Introductory Computer Science Course. *Educ. Sci.* **2024**, *14*, 514. https://doi.org/10.3390/educsci14050514

Academic Editor: João Piedade

Received: 30 January 2024
Revised: 8 March 2024
Accepted: 17 April 2024
Published: 10 May 2024

1. Introduction

There is a well-recognized lack of representation of students and professionals with various social locations within science, technology, engineering, and mathematics (STEM) broadly and more specifically in the computer science discipline. The lack of representation has resulted in efforts to increase the participation and engagement of marginalized and economically disadvantaged students in STEM education [1]. Some of the reasons attributed to the homogenous makeup of computer science classes and in the profession are the lack of early exposure to related concepts, challenges with developing positive science identities, a sense of belonging, and narrow approaches to pedagogy and instruction [2,3]. Much of the efforts to diversify computer science classrooms and the computer science industry have been aligned with relieving the impact of the aforementioned barriers.

Alongside efforts to create heterogeneous educational and workforce environments, there is a risk of characterizing systemic barriers as student pathologies. Curiosities about the attribution of inequities in computer science education (CSE) and the waning participation of students and professionals from marginalized groups have spurred researchers to investigate the issue from sociological perspectives [1,4]. Sociological theories illuminate the importance of considering students' and instructors' identities, agency, and the role of power in society as important elements for informing our understanding of students' experiences and instructor practices in undergraduate computer science education [5,6]. Acknowledging the significance of social locations for creating and maintaining inequities

in CSE guided us to consider ways to "restructure the rules of participation" by providing student-centered curriculum and instruction [1]. Increased participation in diverse gendered enrollment in undergraduate CSE courses has not eradicated racialized and gendered disparities. Despite increases in enrollment due to strategic efforts to diversify CSE classrooms, long-term changes such as those that translate into careers require policy-based, structural changes. Intentional shifts toward equity-focused curriculums and pedagogies can serve as structural changes that support long-term diversity in the computer science field [7].

The authors of this paper are a team of researchers, former ICS students, and post-secondary faculty with expertise in the disciplines of computer science, engineering, cultural studies and learning experience design. For this paper, we designate "Faculty" as the 2 instructors of the current ICS course, and "Researchers" as other post-secondary faculty and the former ICS students that supported the course redesign. Our collaboration was born from a need to address the retention and enrollment of students in an ICS course at a public university located in the midwestern United States. Concerns about enrollment rates (drops, withdrawals, and failures) that mirrored national statistics prompted an examination of student feedback on why they dropped the course [8] and an analysis of the course content and facilitators' instructional methods. Our research team subsequently began the course redesign process and used this paper to reveal findings about faculty responses to the redesign alongside artifacts from the revised curriculum.

Our research includes a theoretical analysis of an existing Introductory Computer Science (ICS) course, a demonstration of how we used the course analysis to inform our ICS course curriculum and instructional redesign, an analysis of faculty responses to the curricular and instructional changes, and examples of the redesigned curriculum. We used the four resource model [9] and critical sensemaking [10] to analyze our collected data (current online curriculum, observational notes from class meetings, transcriptions from faculty meetings, and the redesigned curriculum) and respond to the following research questions:

1. How does the ICS course curriculum (including teaching lectures) align with the four resource model practices of code breaking, text participant, text user, and text analyzer?
2. How can the 4Es heuristic be used to redesign the initial ICS course curriculum and teaching methods to better align with the broad use of the FRM?
3. What are ICS faculty perceptions of the ICS course curriculum redesign process?

In the remainder of this paper, we present the issue of inequitable representation in CSE in extant research literature, describe the theoretical underpinnings that informed our study, and share how we used the 4Es heuristic to redesign an ICS course to center students in their learning along with faculty responses to the curriculum changes.

1.1. Representation and Engagement in Computer Science Education

There is a large body of computer science education research that examines underrepresentation for gendered, economically, and racially diverse students in computer science education [1]. Socially constructed identity markers such as race and gender are connected to students' domain-specific self-efficacy, their experiences of academic exclusion, and their perceptions of individual academic performance and sense of belonging in STEM fields [11]. We find computer science particularly interesting, as women account for more than half of graduates with majors such as chemistry and biology; yet less than 20% for computer science. Cheryan and colleagues [11] identify exclusionary environmental culture, low self-efficacy, and disproportionate early learning of computer science concepts as contributors to the differences in women's engagement with computer science as a major.

Within computer science classrooms, researchers found that female-identifying students' sense of belonging as members of their classroom community was correlated with their perceptions of personal academic and social exclusion [12]. Belonging uncertainty, or concern about the quality of one's social relationships, is of specific concern for students who have intersectional social locations that may include one or more minoritized socially constructed identities such as race and gender [12]. As an example, Black and Latina

women have reported experiences of racism and sexism that led to their feelings of isolation and decreased their sense of belonging within CSE classrooms, resulting in unique and compounded challenges in computer science professions and education [13]. Low sense of belonging and belonging uncertainty are also associated with low program completion rates, which may contribute to the racial and gender gap of women of color in STEM professions [14–16].

A sense of belonging is critically important for the pursuit of computer science education, and students need a developed sense of belonging to pursue careers in STEM education [17]. Thus, efforts to improve diversity within STEM fields have focused on fostering computer science identity among underrepresented student groups. Students agree that engaging in computer science programs designed with intentionality and working with teachers and faculty who are perceived as caring were factors before and during college that contributed to their development of positive science identities [18]. Consequently, there has been a call for educational institutions to systematically improve inclusivity in CSE [18].

Shared language and participation within a discourse community, a knowledge-producing community with shared assumptions, values, and characteristics, can help foster a sense of belonging for computer science students. Thus, instructors must explicitly support students' participation within the discourse community by teaching them the associated literacies and ways of knowing and doing within the discipline while also incorporating students' individual literacies within the discourse community. Disciplinary literacy "emphasizes the specialized knowledge and abilities possessed by those who create, communicate, and use knowledge within each of the disciplines [19]. Explicit teaching of disciplinary literacy, which differs from content area literacy, is promoted in middle and high school science and mathematics education [19]. However, we also recognize the need to attend to disciplinary literacy in post-secondary CSE in ways that affirm and center student identities and experiences.

Disciplinary literacy-focused instruction has been used to change curriculum by offering first-hand/hands-on experiences related to scientific content, eliciting personal experiences that support students' learning of disciplinary-specific content, designing activities for students to practice disciplinary literacies such as debate, and examining how arguments are developed and executed during debates [20]. The 4Es heuristic has been examined as a method to support the teaching of disciplinary literacy in mathematics [21–23], science [24], and other disciplines. Use of the 4Es heuristic in secondary education has also resulted in curricular changes that prompted instructors to merge science and English language arts learning standards, prioritize inquiry and exploration, recognize connections across disciplines, and use student-guided inquiries [19]. Hayden and colleagues [20] found the effects of teaching disciplinary literacy using the 4Es resulted in students' increased abilities to read and analyze science-related literature and to understand main arguments and evidence-based conclusions. Students also improved their abilities to read figures and tables. Instruction grounded in the 4Es provides students with access to the science discourse community, and participation in the discourse community is connected to students' academic engagement and performance [25]. In the following section we discuss the theoretical foundation of our research and provide a more detailed explanation of the 4Es heuristic.

Self-perception and identity are important parts of women's and girls' perceptions of careers in computer science. Akkus and colleagues [26] found that working with middle school girls using a curriculum that combined software programming and identity exploration resulted in improved attitudes toward computer science and increased competence and confidence in their work with computers. Main and colleagues [27] examined the underrepresentation of women in STEM at several different levels (pre-college, college, industry, etc.) and found that the classroom environment, including the design and team/peer interactions, influenced students' decisions to major in computing, and their sense of belonging and self-efficacy were connected to participants' willingness to persist in the computer science majors. High school student dropout rates, coupled with high work-

force needs in STEM and computer science, have spurred interest in student perspectives in higher education. Examining students' expectations about computer science courses and careers is one way that educators can effectively prepare students to complete their coursework and enter the computer science industry.

1.2. Theoretical Perspective

Critical theories are well-suited to address underrepresentation in CSE because of their ability to illuminate power distributions that cause inequitable societal conditions [28,29]. Our research responds to Hubbard Cheuoua's [29] call to action to view issues of participation in computer science courses through a critical lens by using critical sociocultural theory to consider issues of underrepresentation. We view the issue of inequity and participation in CSE from a sociological perspective, which draws our attention to the role of power within systems and structures such as schools and classrooms, and how power distributions may support or constrain curricular and pedagogical revisions. Critical sociocultural theory (CST) holds learning, identity, agency, and power as central and related constructs [30]. According to CST, learning is an act of participation that creates a history for the learner or participant; however, learning is not reduced solely to participation. Learning occurs within the process of participation if students' engagement impresses upon them in a way that creates a history of participation [30]. Learning takes place within a discourse community, a group of people with shared views, practices, and vocabularies/languages [31]. Learning requires gaining access to the discourse community, where students may start as beginners on the periphery and then move closer to being central participants as they hone their expertise. Thus, conditions of learning for students enrolled in computer science courses require their participation and retention of the ways of thinking, knowing, acting, and communicating like computer scientists.

Further, a sociological perspective suggests that full engagement and participation in the CSE discourse community requires students to possess the cultural capital to fully engage in that community and that students' views and dispositions (habitus) may influence successful assimilation into the discourse community [32]. Cultural capital includes the knowledge and skills relevant in CSE, and undergraduate students access cultural capital from activities such as early exposure to computer science concepts and programming exposure in high school. Habitus or students' dispositions toward computer science include stereotypes that students may hold, such as who is/can be a computer scientist. While students' success may be indicative of their possession of and orientations toward capital and habitus, it is the experts, in this case, ICS faculty, who control access to the cultural tools and resources within the discourse community. Access to and control of cultural tools and resources can also be influenced by what Moje and Lewis [30] call "qualities of difference", which are marginalized social identities that result in students of non-dominant race, gender, ability, and social class having less accessibility to the cultural capital required for participation in discourse communities. CST allows us to acknowledge the role of power in creating and maintaining inequities in CSE and guides us to examine how faculty teaching the ICS course supported students' developing disciplinary literacies to participate (and learn) as part of the computer science discourse community. Because disciplinary literacies provide access to the discourse community, we used the four resource model to examine the range of literacy practices that the ICS curriculum supported.

1.3. Four Resource Model

We sought to analyze the existing ICS course prior to making changes to its curriculum. Thus, we needed a framework to help us understand the breadth of literacy practices that faculty used to support students' disciplinary literacy development. We referenced Luke and Freebody's [9] four resource model (FRM) for the task of analyzing the pre-existing ICS course curriculum. The FRM is a theoretical perspective that describes how readers engage with texts, where reading is broadly defined as meaning-making and texts are not limited to print but may also include visual images and the world around us [33]. The FRM is a useful

approach for understanding how students in the current ICS course access disciplinary literacies through a set of resources that represent a range of reading practices that students draw upon when developing disciplinary literacies [34]. The FRM perspective highlights four key activities produced by proficient readers (called resources); they are codebreaker, text user, text participant, and text analyst. Codebreaking refers to the understanding of the conventions, signs, and symbols of how the text is constructed; it is also called decoding the language. Text participants can make meaning (literal and inferential) from a text by differentiating it from others, and describing its salient characteristics as they relate to computer science. Text users can harness their knowledge of a text to use it appropriately; communicating within the context of the discipline. Text analysts recognize the text as being associated with power and question how the text may act to position them; this is a critical form of engagement that empowers students to move beyond blind consumption of disciplinary knowledge [35].

Codebreaking, text user, text participant, and text analyst are interdependent resources that readers do not follow in a specific sequence during the reading process or during the developmental stages of learning to read texts [36]. However, the four resources vary in how deeply readers understand, access, and engage with texts. The FRM helped us understand if and how the ICS course curriculum and teaching methods positioned students to engage with texts in the ways necessary to support the breadth of practice required for disciplinary literacy. After our analysis of the ICS course curriculum, we sought to expand students' opportunities to develop disciplinary literacies and used the 4Es heuristic to support our course redesign.

1.4. 4Es Heuristic

The 4Es heuristic (engage, elicit/engineer, examine, and evaluate) is grounded in critical sociocultural theory and related to disciplinary literacy. It is a method for (re)designing curriculum and instructional content that contextualizes the required cultural capital for success in CSE within the scope of students' lived experiences. The 4Es heuristic is designed for teaching disciplinary literacy, where 'literacy' refers to the mastery of not only the content of a discipline but also the breadth of cultural capital required to fully participate in the discourse community of a CSE course. The 4Es also guide instructors to support students as they form their habitus and expand their views of computer science overall (including who can be a computer scientist). The 4Es represent teaching practices that instructors may use to support students drawing upon their existing cultural capital to engage in the computer science discourse community. The heuristic is organized by four directives that guide instructors to support students' full engagement with both disciplinary content and disciplinary literacy, which serve as capital within the discourse community and include the ways of being, doing, and knowing computer science. The 4Es guide instructors to do the following:

1. ***Engage*** with the practices of the discipline as part of students' daily routine, including the vocabulary, habits of practice, etc.
2. ***Elicit/engineer*** opportunities for students to engage in disciplinary practices and the discourse community. Instructors should first elicit and consider students' current skills as they engineer experiences for students to gain new knowledge.
3. ***Examine*** reminds instructors to guide students to closely examine words, such as disciplinary-specific vocabulary.
4. ***Evaluate*** has instructors guide students through a meta-discursive process by asking themselves when, where, and why disciplinary language is useful.

Working with students through the 4Es heuristic positions instructors as responsible parties for uncovering a hidden curriculum within discourse communities; thus providing access to a wider range of perspectives and practices from those who are typically marginalized from the discourse community due to qualities of difference, such as gender, or less accessibility to the cultural capital required for success in existing CSE courses.

Faculty instructor feedback was also a critical element in our research. We used critical sensemaking to understand how faculty responded to curriculum shifts.

1.5. Critical Sensemaking

The critical sensemaking framework (CSM) was designed to understand how minoritized undergraduate students made meaning of their campus climate [10]. Within this study, we utilized CSM to explore how faculty make sense of implementing a curriculum that centers students. CSM is grounded in the work of Weick [37], which determined seven components that are engaged as one constructs meaning or understanding. According to Weick [37], the sensemaking process occurs when an individual is faced with an experience that creates uncertainty (new curriculum). The framework outlines that meaning-making is a ***social*** construction that includes one's own experience and the requirements of their environment [38]. Sensemaking is also cultivated through how an individual perceives oneself (***identity***) and their ability to both experience and influence their environments (***enactment***) [39]. As individuals engage in new experiences, previous events in their lives also inform how they interpret the novel encounter (**retrospection**). The process of sensemaking is continuous (**ongoing**) [39]; as new situations arise, one is constantly identifying information to inform their perception of it being applicable (**extraction of cues**) [40] and reasonable given their previous experiences and beliefs (**plausibility**) [9].

Despite Weick's attempt to identify and define the tenants of sensemaking, one major critique is the lack of consideration for power within the process [38]. Given the role of instructors in deciding curriculum materials and learning experiences, while situated within the requirements of the institution as faculty members, there are systems of power at play within this dynamic that shape their perspective and subsequent sensemaking [39]. "CSM builds on the foundational notions [of sensemaking] by considering the role of power—especially power as exerted by context, structure, and discourse" [10], p.87. Schildt and colleagues [38] discuss that power is not solely an act being imposed on another; it may derive from historical discourse and can exist both through deliberate action and historical structures that have upheld perceived power, though not explicitly stated. Thus, CSM allows us to explore the discursive elements involved in faculty sensemaking while acknowledging the role of power in their meaning-making process.

2. Materials and Methods

We conducted a theoretical analysis of an existing ICS undergraduate course to examine the literacy practices and understand how students are positioned in relation to the discourse community, then we used the 4Es heuristic to redesign a portion of the course, and finally we examined faculty feedback on the redesigned course. The 4Es heuristic provided a framework for us to restructure the course in a way that foregrounds the significance of disciplinary literacy and access to the discourse community. In this section, we describe the research participants, the ICS course that was our subject, our strategy for redesigning the curriculum, and the perspectives of faculty instructors on the process.

2.1. Context

2.1.1. Research Participants

The initial ICS course analysis and redesign were completed by the co-authors of this paper, which included faculty instructors of the ICS course and other members of the research team. The two faculty instructors of the ICS course and the other members of the research team also contributed to the conversations that were recorded, transcribed, and analyzed to better understand faculty perspectives on the curriculum changes. Both faculty instructors have taught the ICS course for more than two years at their current university.

2.1.2. Course Description

We conducted the analysis and curriculum redesign for an ICS course at a large midwestern Research 1 institution. The course was designed for any student seeking

to learn beginning computer science concepts and possibly persist into the major; high school math was the only course prerequisite. Students learn a functional programming language that covers arithmetic, definitions, design recipes, enumerations, and other common introductory topics. The course has been offered for almost two decades, with some variations made by faculty to support student success. Prior to the redesign, the ICS course followed a flipped classroom format. Students were required to view video lectures and complete short exercises before attending class. During the 75 min in-person class sessions, the instructor provided additional details related to the video lecture content, and students completed additional in-class exercises. Graduate or upper-level undergraduate students facilitated weekly, mandatory laboratory sessions where students were required to complete assignments that supported their continued engagement with the content covered during weekly instruction. To support engagement outside of the scheduled instructional time and offer just-in-time assistance, the instructors incorporated an instant message platform for students to ask questions and engage with their peers.

In a previous study data regarding drop-out rates for the ICS course was examined for three consecutive semesters (fall 2021, winter 2022, and fall 2022) [8]. The results revealed that undeclared students dropped the course at a significantly higher rate compared to those who were declared CS majors. Further, women drop out of the course at a significant rate compared to their male counterparts. There was no significant difference in the drop-out rate based on race. However, minoritized students only made up approximately twenty percent of those enrolled in the ICS course. As well, the institution being examined requires students to provide an open-ended reason for dropping a course. An analysis of the drop-out reasons over the three semesters revealed the most common reasons for dropping a course could be categorized as "academic struggles", "dislike/not required", "not prepared". We drew on the results of this previous study to inform the curriculum redesign.

2.2. Analysis

2.2.1. ICS Course

We conducted a content analysis of the curriculum materials within the original ICS course. The original course materials were housed within an online learning management system and organized by the corresponding lecture (e.g., Lecture 1). There were a total of 18 lectures in the course. Each lecture folder consisted of recorded videos of the faculty explaining the topic for the week and demonstrating how to complete the subsequent activities. There were multiple videos within each lecture folder, and a number of exercises were to be completed after watching the video(s). We followed the method of analysis outlined by Caro and colleagues to code each video and the accompanying exercises for how they supported and/or activated certain learner roles identified in the FRM [41]. Two researchers from the design team reviewed all the assignments and lectures that were assigned for the first two weeks of the course. Similar to Caro and colleagues, each item was categorized according to the resource/role it supported learners in performing while completing the learning experience, and more than one category could be identified within a single activity [41]. For example, when students were asked to write down data definitions for structures to identify their functions, they were coded as ***codebreakers***. Also, when learners were instructed to create two new data definitions, they were labeled as ***content users***. When learners were asked to look for non-essential differences in data definitions, they were labeled as ***content participants***. When learners were asked to share three things they would advise future students, they were coded as ***content analyzers***.

2.2.2. Curriculum Redesign Strategy

Big Ideas.

Our goal for the curriculum redesign was to create a more student-centered approach to teaching in the ICS course that would engage students' cultural capital, and provide them with explicit opportunities to learn the disciplinary content and literacy required for engagement in the discourse community. Stevens and colleagues [42] define "Big Ideas"

as the central and fundamental aspects of a discipline that lead to students' sustained understanding and progress. We began the curriculum redesign by working collaboratively to identify the five big ideas that would support students' mastery of computer science's disciplinary content.

Next, we organized all of the course content around the five big ideas, ensuring that each topic was directly connected to one of these core concepts. We then developed learning goals (specific to this course) to reflect the desired conceptual understanding that students should have as a result of engaging in related learning experiences and specified the prerequisite knowledge for learners to successfully master the learning goal. The following is an example of one big idea, followed by the learning goal and prerequisite knowledge:

Big Idea

Computer programs that work with similar types of data follow similar patterns in how they are written. As learners practice more and more computer science they learn to recognize these patterns and then apply them.

Learning Goal

By the end of the course, students should be comfortable working with a variety of built-in and custom data types/structures. Often, the types of input and output required for a function suggest an appropriate template to use, only with a couple of small details left to fill in. By learning to recognize similarities and common patterns between functions, students will be able to quickly orient themselves when writing new functions. This will also speed up debugging, as students can note whether they made an error with the overall template/flow of their code or with a specific detail.

Prerequisite Knowledge

(a) Pattern recognition—being able to notice commonalities and make generalizations. (b) In contrast, being able to identify essential differences between similar things.

Finally, we listed potential difficulties and misconceptions related to each big idea. As an example, students commonly attempt to combine data types, which often leads to error as data types follow specific and uncompromising rules. Breaking the rules of a data type may result in breaking a function.

After outlining the big ideas of computer science and using them to align the learning objectives, we identified the prerequisite skills, not to qualify or disqualify students but to assist instructors with locating the source of misconceptions and identifying potential difficulties. After outlining the full picture of the course, we turned our attention to the 4E's heuristic as a method to address our concerns about centering student identities and broadening literacy participation to develop disciplinary literacy and increase their opportunities to participate in the discourse community.

4E's Heuristic.

We utilized Moje's [25] 4Es heuristic to guide our development of weekly lesson plans that would support students' identity, agency, and power within the community of practice while sufficiently supporting their content knowledge of the domain. The 4Es heuristic appealed to us as a design strategy because it is descriptive, but not prescriptive. Using it as a framework provoked faculty/instructors to consider ways to prioritize student experiences and identities while offering flexibility for individual instructor preferences and expertise. The following represents how we engaged the elements of the 4Es to accomplish our redesign goals:

1. Engage helped us support students in exploring the "everyday practices" of the discipline of computer science. We utilized the problem-framing component to support learners' motivation.
2. Elicit/engineer helped us support beginning computer science students, especially those in their first year of college and those with varied previous experiences, by emphasizing the meta-strategies that will elicit the thinking necessary for success in the course, and computer science work in the future.
3. Examine guided us to ensure that all students can access the course content regardless of their level of prior knowledge by questioning all terms, including how and why

they are used. This also allows instructors to assess the schema and preconceptions students are bringing into the course.

4. Evaluate guided us to provide students with the opportunity to challenge concepts and explanations that are bogged down in jargon and unnecessary complexities. If a term or concept is difficult to refine without referencing a prior assumption/understanding, then students are encouraged to interrogate why that might be the case.

Pacing Guide.

The pacing guide was an important artifact from the redesign that incorporated what we learned from organizing the course content around the five big ideas and using the 4Es heuristic to support students in developing their disciplinary literacy. The pacing guide outlined the semester by week, where each week instructors taught based upon one of the five big ideas. We described the weekly concepts to be covered, delineating the disciplinary concepts from the disciplinary literacy practices. We did this to ensure that instructors can explicitly identify how they are responding to the call from the 4Es (centering student experiences and identities) as they teach the content. The pacing guide also includes a weekly note to students that explains why that week's content is important for their domain knowledge.

2.2.3. Faculty Responses

We applied ordered network analysis to our faculty response data using the ONA Web Tool (version 1.7.0) [43,44]. Our units were divided into two groups: the faculty and the research team, and we analyzed the conversation using a moving stanza window of four lines. The resulting directed networks are aggregated for all lines for each unit of analysis in the model. We created a model that represents aggregated networks using a binary summation in which the networks for a given line reflect the presence or absence of the co-occurrence of each pair of codes. Our ONA model included the following codes outlined in the CSM framework: identity, social, plausibility and retrospection. Some codes accepting, student learning, curriculum, pedagogy, curriculum, struggling, and theoretical were not outlined in the CSM framework but added to the code book to account for their occurrence in the discourse. We defined conversations as all lines of data associated with group discussion on curriculum design. The ONA model normalized the networks for all units of analysis before they were subjected to a dimensional reduction, which accounts for the fact that different units of analysis may have different numbers of coded lines in the data. For the dimensional reduction, we used a singular value decomposition, which produces orthogonal dimensions that maximize the variance explained by each dimension [45,46]. We visualized networks using directed network graphs where nodes correspond to the codes and edges reflect the relative frequency of co-occurrence, or connection, and connection direction between two codes. The result is two coordinated representations for each unit of analysis: (1) a plotted point, which represents the location of that unit's network in the low-dimensional projected space, and (2) a directed weighted network graph. The positions of the network graph nodes are fixed, and those positions are determined by an optimization routine that minimizes the difference between the plotted points and their corresponding network centroids. Because of this co-registration of network graphs and projected space, the positions of the network graph nodes—and the connections they define—can be used to interpret the dimensions of the projected space and explain the positions of plotted points in the space. Our model had co-registration correlations of 1 (Pearson) and 1 (Spearman) for the first dimension and co-registration correlations of 1 (Pearson) and 1 (Spearman) for the second. These measures indicate that there is a strong goodness of fit between the visualization and the original model.

ONA can be used to compare units of analysis in terms of their plotted point positions, individual networks, mean plotted point positions, and mean networks, which average the connection weights and directions across individual networks. Networks may also be compared using network difference graphs. These graphs are calculated by subtracting the

weight and direction of each connection in one network from the corresponding connections in another. To test for differences between the networks, we applied a Mann–Whitney test to the location of points in the projected ENA space for units in the faculty and research team.

3. Results

Our first research question examined how the initial ICS curriculum and teaching lectures aligned with the FRM practices of codebreaking, text participant, text user, and text analyzer. The initial ICS course curriculum dedicated less than 40% over the course of the 18 weeks to breaking down the basic components of syntax and grammar in computer science-related language and practices. Instructors spent more time/more of the course content with foundational elements of codebreaking later in the course than at the beginning. The learning experiences provided more frequent opportunities for students to be guided/supported in making meaning from course content, but the majority positioned them as text users and they very rarely were guided to engage with course content as text analysts. Figure 1 represents the distribution of resources that instructors addressed to students in the ICS curriculum.

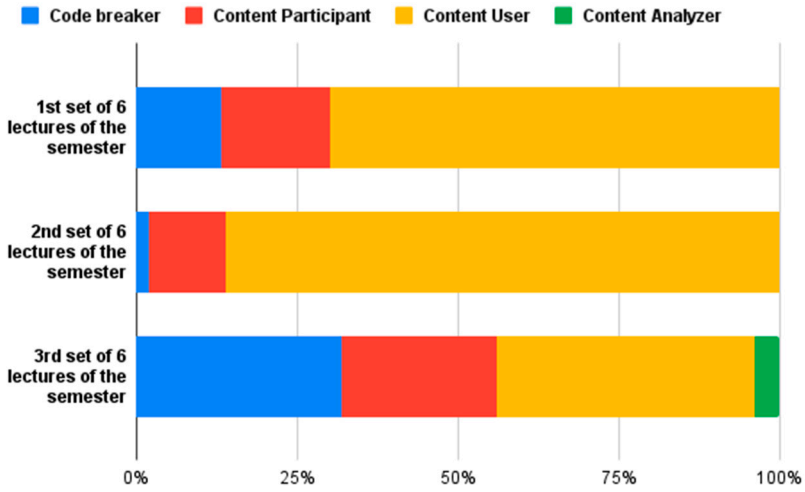

Figure 1. The content analysis results from the original ICS course. The semester was divided into three sets of 6 weeks in chronological order, for a total of 18 weeks. Each set illustrates the frequency with which one of the four resources/learner roles (codebreaker, text/content participant, text/content user, and text/content analyzer) was supported within a learning activity.

For question two, we wanted to explore how the 4Es heuristic could be used to redesign the ICS curriculum and teaching methods to better align with the more comprehensive use of the resources named in the FRM. We used the 4Es heuristic to support the reimagining of the ICS course in the following manner: During the first week of the course learners, are engaged in the "everyday practices" of the discipline (Figure 2). This begins by contextualizing the work for the week within a problem or overarching question. For week one, the topic of rocket science is used to contextualize the material for students. They are asked to think of the many pieces that are required to successfully launch a rocket. Items such as the *design* would be discussed since the rocket has to be able to propel itself with incredible force and accuracy, survive exiting and re-entering Earth's atmosphere, and be livable for the astronauts. Students would recognize that the *construction* (e.g., using proper materials, being precise in measuring and welding) of the rocket would be important in its launch. Training could be recognized as a vital component of a rocket launch, as astronauts and ground controllers have to be versed in a variety of procedures and prepared for all scenarios (Apollo 11 vs. Apollo 13). Additionally, *calculations* would be necessary to determine how much fuel is required, what path the rocket should take to reach its destina-

tion, and how gravitational forces or weather will impact its trajectory. Students would discuss the importance of *communication*, as constant interaction and engagement have to be maintained between ground control and the astronauts, even as the latter are in space. Contextualized within the overarching question, students are provided the opportunity to engage with data that will encourage them to identify patterns that appear within the data. As well as salient features of the data that will be most relevant to their analysis. The goal would be for students to identify patterns and ways that data can be grouped and interpreted.

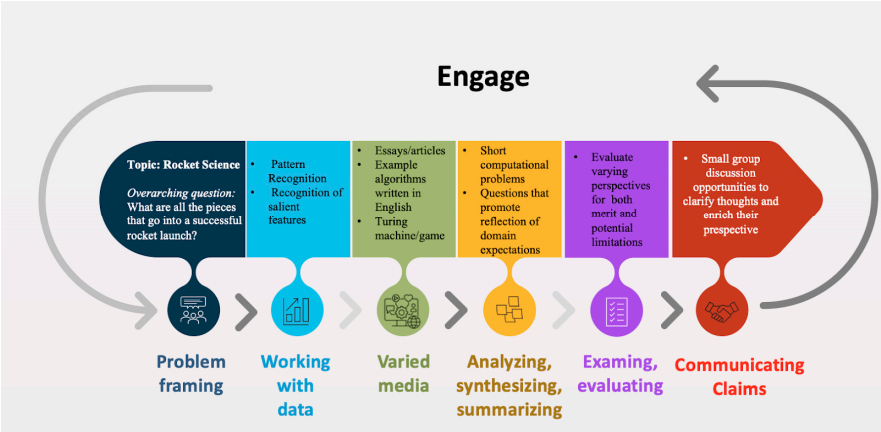

Figure 2. This is a revised figure of Moje's (2015) 4E's heuristic figure that explores how "engage" is achieved through the disciplinary cycle. This figure includes examples of how each aspect of the disciplinary cycle can be used to support ICS students' disciplinary literacy.

Continuing to build students' identities and beliefs, we expand their sense of what it means to do computer science beyond coding. This includes showing them what an algorithm looks like in words versus a coding language or a physical simulation utilizing a Turing machine. To build students' agency and redistribute power within the learning experience, we provided learners the opportunity to reflect on their understanding of what the field of computer science entailed and to examine various perspectives and evaluate them for both their merit and potential limitations. Week one's learning experiences would culminate in a discussion of their position in both small and large groups. This formative assessment technique positions both the instructor and the student as learning resources.

Elicit/engineering within the course redesign is nurtured as students are allowed to experience computer science beyond coding. The design makes no assumptions about the level of prior experience students bring to the course. The focus in week one and throughout the course is on developing the meta-strategies that will elicit the thinking necessary for success in this course, and computer science work in the future. Within the course redesign, we acknowledge that students are typically in their first or second semester of college and require more support and scaffolding as they engage with the disciplinary practices (Figure 2). The practice of *examining* is nurtured within the learning experience as terms such as computers and science are called into question. The discourse within the course is focused on centering students' voices in lieu of lecturing. For example, in a large group setting, students completed a mind map for a concept in computer science. The map consisted of purely students' suggestions, with the instructor providing feedback only at students' request. This type of formative assessment can identify what schema preconceptions are informing students' conceptions. The course redesign assists students in examining the ways of thinking and knowing in computer science. Throughout the learning experience, students (and teachers) are encouraged to redefine definitions and explanations to their essence, so that they are free of jargon and overcomplication. If a concept or term is deemed too complex and cannot be simplified or explained without

reference to a prior assumption/understanding, students are to interrogate why that might be the case.

There were two key changes that allowed for shifts in the curriculum toward identity and power. The first key change was the development of big ideas for computer science. In addition, we identified why each big idea was important and what students should learn in the specific course related to the big idea. With each big idea, we list the types of previous knowledge we deem necessary to support successfully mastering the discussed concept, along with potential student difficulties and misconceptions.

BIG IDEA 1: Computer programs are modular, combining simple pieces into a complex whole. Similarly, difficult problems in computer science can be solved by breaking them down into easier subproblems.

Significance: Some problems can look intimidating, but breaking them down into linear steps helps to make sure the problem is solved correctly. Throughout their careers, the big, complicated things that people create could be made by less experienced individuals, but they do not know that the complex codes or problems are really a combination of smaller, more attainable problems. If you are working with a larger team, they are working on separate tasks, which are then all put together. So, working on smaller parts of a whole is common in CS.

What students should learn: By the end of C211, students should be able to work through multi-step coding problems by decomposing them, building smaller programs to perform subtasks, and then combining these programs logically. The course will start with teaching how to write these simple programs, often consisting of functions that take a small number of inputs and produce a single output, and variables that are defined as the application of a single function. Once students are comfortable working with these basic building blocks, the course will teach them how to combine them in increasingly sophisticated ways, such as function nesting, conditionals, helper functions, recursion, and abstraction. Emphasizing this approach will prepare students for the modularity inherent in computer science.

Specific prerequisite knowledge:

- Algebra (especially working with functions);
- Vague familiarity with modular design.

Potential student difficulties and misconceptions:

- A common difficulty is trying to tackle the entire problem at once, instead of taking it one piece at a time;
- Not checking that every part works before combining them, then being unable to identify the problem when the whole fails;
- The misconception that all coding questions are solved in one function; this often leads to struggle as there are many times where a solution involves a helper function, which makes the question much easier;
- Combining functions arbitrarily rather than reasoning about how data should flow through them (often helps to use step-by-step breakdowns).

BIG IDEA 2: Code should be as efficient as possible, optimized for time, memory, and storage space.

Significance: Human usability is important. Time matters in how long it takes to execute a task. If a program takes up too much hard drive space, then computers will not be able to store it. This topic is less talked about, but optimization is necessary later on. Doing something by hand vs. on a computer to emphasize optimization.

What students should learn: While optimization is explored more thoroughly in later computer science courses, ICS students should be able to explain why time and memory efficiency are important programming tenets. Fast and slow algorithms (e.g., for sorting and exponentiation) can be demonstrated and compared, illustrating how increasing the magnitude of the input can dramatically increase the runtime of a program, especially if optimization strategies are not taken. Real-world scenarios (e.g., a search engine in a

hospital database, real-time GPS navigation) should also be explored to underline the importance of this concept.

Specific prerequisite knowledge:

- Having been on the user end of computer programs;
- Experiencing fast and slow runtimes (motivates the big idea).

Potential student difficulties and misconceptions:

- The misconception that a single problem only has a single solution—a coding question has countless different solutions, each one varying in its speed and memory usage;
- Not taking these principles to heart when they start learning to code, then eventually reaching a point where they struggle with code efficiency.

BIG IDEA 3: Computer programs that work with similar types of data follow similar patterns in how they are written. As you practice more and more computer science, you learn to recognize these patterns and then apply them.

Significance: Help students identify commonalities, reduce coding time, and support debugging.

What students should learn: By the end of ICS, students should be comfortable working with a variety of built-in and custom data types/structures. Often, the types of input and output required for a function suggest an appropriate template to use, only with a couple of small details left to fill in. By learning to recognize similarities and common patterns between functions, students will be able to quickly orient themselves when writing new functions. This will also speed up debugging, as students can note whether they made an error with the overall template/flow of their code or with a specific detail.

Specific prerequisite knowledge:

- Pattern recognition—being able to notice commonalities and make generalizations;
- On the flip side, being able to identify essential differences between similar things.

Potential student difficulties and misconceptions:

- A very common difficulty students experience is trying to mix data types together, which often leads to an error, as data types follow very strict rules that break an entire function when not followed (using a function that takes in a number but giving it a string);
- Not knowing/following the usual template for a function—making one up on the spot is usually going to be incorrect;
- The misconception that a computer knows exactly what you want it to do—just because you understand the purpose of your code does not mean the computer does;
- Trying to solve the problem before analyzing it and figuring out what pattern the code should follow.

BIG IDEA 4: Writing computer programs requires lots of testing to figure out what does and does not work and to account for all use cases.

Significance: One of the core things in computer science is not how you solve it but how you break it (you are not designing your problem for yourself but for other people— a lot of people do not know how to use computers perfectly, so write code as if the person is not going to use it perfectly) and then how you will patch up those possible breaks. If you have a hypothesis and run experiments to test the hypothesis, your example should cover all inputs. Being willing to write and rewrite your code until it works. Supports failing and retesting.

What students should learn: After ICS, students should be able to identify both the regular uses and edge cases within a standard programming question. There are several common edge cases that can be identified, such as using negative numbers or using incompatible data types, and these should be preemptively dealt with by students using tests/examples. After covering the common edge cases, students should also be able to identify more subtle errors that their code may not account for and effectively patch them while maintaining a solid code structure (ex: not just adding a conditional that covers a single error, but rather dealing with the root of the error itself). In debugging,

students should be willing to try a variety of solutions to get their code working properly, experimenting with what does and does not work—this process is essential in all kinds of computer science.

Specific prerequisite knowledge:

- Familiarity with the scientific/empirical method (hypothesis -> experiment);
- Previous experience with detail-oriented work.

Potential student difficulties and misconceptions:

- Writing tests that fit the result of your code, rather than writing code that fits the result of your tests;
- Not writing enough tests/not writing any tests;
- Immediately giving up when the code does not work instead of first trying some plausible fixes;
- When a test does not pass, tweak the code so it now passes that test instead of actually analyzing why the test did not pass;
- Not doing step-by-step breakdowns for one or more example inputs, especially for more sophisticated problems;
- Believing that if all the tests they've written pass, their function must be working properly (there could be edge cases and other types of input that they have not considered and tested).

BIG IDEA 5: Code should be precise and detailed to be understood by the computer and well-annotated to be understood by the student and other programmers.

Why this is important: Knowing what your functions do is important, especially when you have more than one. Focus on clarity. Part of annotation is choosing good variable and function names. It is important for not only how everyone else understands your code but also for how you understand your code. Computers do not automatically understand the purpose of code—they need everything spelled out in detail in order to work properly.

What students should learn: ICS students should learn the proper foundations to write well-formatted and legible code that can be understood by themselves, the computer, and other programmers. This includes choosing good variable and function names, as those can help the student figure out how elements should be used or combined as they write their code, as well as proper indentation to illustrate the flow of the code. Someone else should be able to view their code, and although they may not be able to know its exact functionality, they should be able to recognize the inputs that it deals with and have a vague idea of its purpose. It is also much easier to spot errors and bugs when the program is written cleanly; otherwise, simple mistakes in punctuation or structure can be masked by the "noise" of poor formatting.

Specific prerequisite knowledge:

- Background in English writing (e.g., proper grammar, punctuation, and formatting);
- Experience with miscommunication—how has imprecision/ambiguity led to misunderstandings in social interactions?

Potential student difficulties and misconceptions:

- The misconception that code should be written in a way that only they can understand. Code clarity means that anyone should be able to read and roughly understand its purpose;
- The misconception that having the code work properly is good enough, even if their solution is indecipherable;
- Random/undescriptive variable and function names (x, var1, blah, etc.);
- Poor indentation that leads to confusion about the structure of the code;
- Not keeping track of what kinds of inputs and outputs functions take;
- Not keeping track of the purposes of individual functions or what certain variables mean;
- Getting into bad habits early—as the course progresses and the problems become more challenging, the sloppiness of the code will catch up to them.

The second key change was the development of a weekly pacing guide for the entire semester-long course (Figure 3). Each week, identify which big idea will be covered to assure alignment with the major ways of thinking and knowing within the discipline. The facts and concepts that will be covered that week are outlined in the pacing guide. Instructors are provided with examples of problems they can provide students with to apply their knowledge. The pacing guide is designed for both students and instructors to view as a way to communicate the expectations of the instructors to the students. As such, it includes skills that students should be able to demonstrate at the end of each week, as well as types of projects/assignments that will support students in demonstrating mastery of the concepts. Each week contains a section that identifies what knowledge from previous weeks students are expected to engage in order to master the concept or support their disciplinary literacy. The pacing guide also includes a weekly note explaining why the concepts for that week are important both within the course and the discipline.

Week	*Big Ideas*	Facts and concepts	Problems to apply knowledge	Skill to demonstrate	Project/assignment to show mastery	What knowledge from the previous week are we engaging?	Why is this important?
1	*1a, 1b, 2*	What is computer science, computational thinking, modular design, common problems in CS	Breaking down a system/problem into constituent parts (e.g., how many types of employees needed at a restaurant and their individual functions), then breaking down a common piece of software in a similar fashion	Thinking like a computer scientist *before* learning to code	Short written responses about what the student thinks computer science is, and what they hope to learn from the course and their studies at IU	This should not be building on anything. This week should be understandable by students with some or no experience with coding or computer science.	If you (the student) are interested in computer science, this course should inform you right off the bat what the field looks like, and begin teaching you how to think like a computer scientist. Though you may be anxious to start programming immediately, it's important to first get an idea of the types of thinking and learning that will be engaged in this course and future studies at IU, and see if it's a good fit for you.
2	*5a,5b,3a*	Introducing Racket (or other preferred language), good coding etiquette, simple data types, what is a variable and what is a function	Analyzing examples of good and bad code formatting & suggesting edits, constructing simple images, arithmetic operations on numbers, identifying/differentiating variables and functions	Processing of basic data (numbers, images, strings), proper formatting & annotation, explaining need for variables and functions	Problem set: series of short code exercises defining simple variables and functions, graded for completion, accuracy, and cleanness/legibility	Applies the computational thinking explored in Week 1 to begin working with real code, creating the smallest "building blocks" that will continue to be elaborated on and combined throughout the course, also laying foundations of good code etiquette	Writing computer programs (coding) is an incredibly important and common skill within CS and a variety of related career paths. It also enables a hands-on approach to learning and applying CS concepts. Though this week will focus on the basics of Racket, e.g., defining straightforward variables and functions, starting simple will lay the foundations for you to write functional and legible code from here on out.
3	*4a, 4b*	Empirical approach (being willing to experiment with code and solution methods), testing for all use cases, conditionals, enumerations	Making examples of incorrect code work properly, coming up with tests that cover all types of input, producing enumerations, writing conditional templates for given enumerations	Willingness to experiment and carefully revise, accounting for all possible use cases, handling conditionals & enumerations	Problem set: making a series of conditional handlers for simple games (e.g., two different outcomes for a coin toss, six for a die roll, twelve for two dice)	Explores the testing aspect of computational thinking (Week 1) and the beginnings of modular design, combining the simple code blocks of Week 2 according to conditional logic	The more complex a program is, the more possible ways there are for it to fail. These can range from minor inconveniences (e.g., a graphic doesn't load properly) to major disasters (e.g., a hospital database crashes and corrupts all patient records). The only way to make sure that your code works is to test it thoroughly, accounting for all use cases, and then revise it if necessary. It's also helpful to make sure that individual

Figure 3. Illustration of the pacing guide for the first 3 weeks of the redesigned course.

Our third research question required us to examine the ICS course instructional faculty's perceptions of the redesign process. Figure 4 illustrates that along the X axis (SVD1), a Mann–Whitney test showed that the faculty (Mdn = −0.47, N = 2) was not statistically significantly different at the alpha = 0.05 level from the research team (Mdn = 0.24, N = 3, U = 0, $p = 0.20$, r = 1.00). Additionally, we found that the faculty (Mdn = 0.23, N = 2) was not statistically significantly different on the Y axis (SVD2) at the alpha = 0.05 level from the research team (Mdn = −0.39, N = 3, U = 4.00, $p = 0.80$, r = −0.33). Although the networks were not statistically significant, there are differences highlighted by the connections within the networks. For the faculty network, identity played the largest role in their sensemaking during the conversation. When looking at the connections to identity, student learning, curriculum, and struggling had the largest ties to it. This highlights that the faculty were struggling to find the connection between their own identity and practices that help students learn in the classroom.

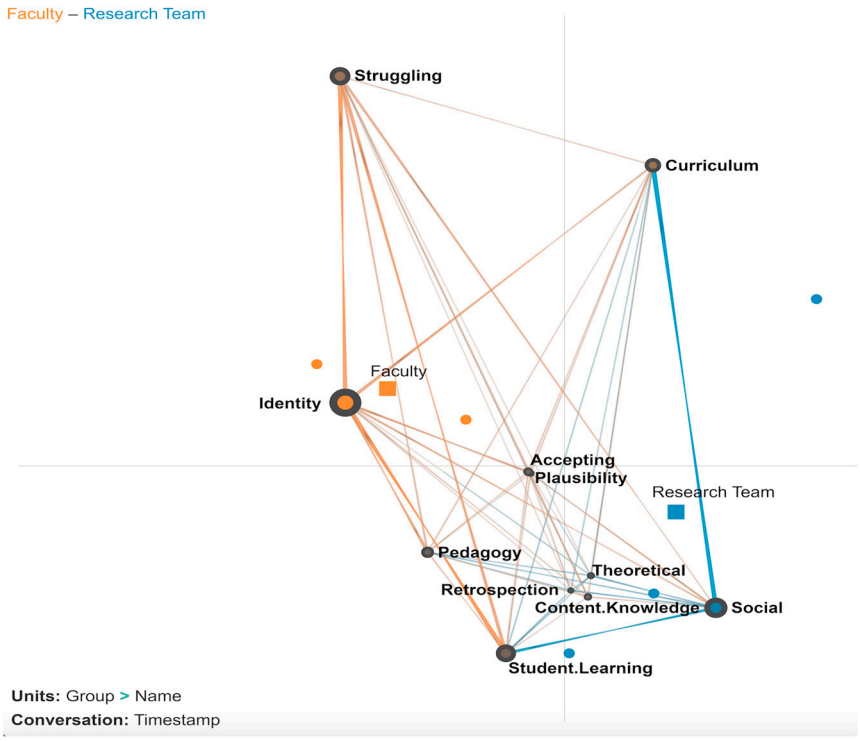

Figure 4. The model represents the discussion between the research team (blue lines) and the faculty team (orange lines). The model indicates the co-occurrence of codes derived from the CMS framework. The size of the nodes (circles) indicates the number of occurrences of the code during the discussion. The thickness of the lines indicates more co-occurrences of the two codes. The color of the lines and nodes represents the group (research team or faculty team) it is most represented by.

For the research team, the social aspect of sensemaking played a large role in their network, with the largest connections to curriculum and student learning. These connections show the conversation required the research team to provide broad group conversations on practices to help students learn in the classroom. When looking across the networks, we can see that the research team was continually helping the faculty through their struggle with student learning and curriculum, but the discussion did not shift towards the faculty's acceptance of the proposed curriculum (low instances of accepting and plausibility codes). Interestingly, despite its central role in the faculty network, there was little connection to identity for the research team.

4. Discussion

There is a need to address the lack of gender, ethnic, and experiential diversity among students in computer science education courses. Efforts to solve this issue have previously focused on increasing students' content knowledge by providing them with early exposure to computer science concepts and increasing experiential learning opportunities. While these methods have been marginally effective, they do not address the issue of inequity from critical perspectives that can illuminate structural barriers that limit access to power, such as students' limited access to computer science discourse communities. There is also the danger of pathologizing student performance in ways that label students as underperforming and underachieving without examining the course curriculum and instructor pedagogy. One way to broaden students' accessibility and to diversify perspectives within the computer science discourse community is to support students' disciplinary literacy development. Learning the ways of knowing and the common language and practices used in computer science education and industry affords students from varying backgrounds greater opportunities to understand, engage with, and critique the discipline, thus diversifying perspectives and growing the field.

Disciplinary literacy development is a process where students practice making meaning of the words, habits, and practices of computer science as a discipline. Students may investigate what terms and vocabulary are used and why, learn how to appropriately use related equipment and tools, observe and practice interpersonal communications within the computer science industry, use their knowledge to execute tasks, and critique existing processes and procedures to push the current boundaries and bring new ideas into the field. The four resource model represents a range of literacy practices that occur when students engage in meaning-making. We used the FRM as a framework for analyzing the initial ICS course to understand how the curriculum aligned with the framework and if students were provided with opportunities to practice the range of meaning-making processes (from codebreaking to critiquing) that are required for developing disciplinary literacy and engaging in the computer science discourse community.

Students were positioned in the curriculum most often as text or content users because they were provided with a list of instructions for executing functions with very little background on the origin of the concepts (codebreaking), how to make meaning of the language/vocabulary in multiple ways related to computer science (text participant), and encouragement to challenge the status quo of how and why processes are executed in the discipline. Limiting students' opportunities to deconstruct discipline-specific language through codebreaking, their literal and figurative meaning-making through text and content participation, and their critical thinking through text and content analysis is problematic, in part because it limits their full participation in the CSE discourse community. Although interventions have focused on approaches that mostly address gendered inequities in CSE, research has demonstrated the need to broaden, more generally, perspectives on computer science and computer scientists [47]. Curriculum that foregrounds disciplinary literacy, specifically through the 4Es heuristic, provides opportunities for diverse students and their perspectives to participate in widening CSE perspectives. This may include students revising and challenging longstanding ideas about the field of computer science and participating in the full range of literacy practices identified through the FRM.

We found the 4Es heuristic to be most useful in aligning with the FRM framework when we focused on engagement, eliciting/engineering, and evaluation. We designed opportunities for instructors to *elicit and engineer* opportunities for students to participate in computer science education in ways that draw upon their prior knowledge and understandings. This aligned well with codebreaking because it does not assume all students have prerequisite knowledge and builds capacity for broadening their schema from their current understandings. When using *engagement* as a design principle from the 4Es heuristic, we were guided to include hands-on opportunities for students to explore and experience concepts reflected in the ICS course content, which we believe will support their practices as text or content users. Students are guided in the curriculum redesign to take a group of derived patient data and identify patterns and ways that the data can be grouped and interpreted. We used *examination* from the 4Es heuristic to guide instructors to support students' in paying close attention to domain-specific language and practices, something that is required across the breadth of meaning-making processes in the FRM. Finally, *evaluation* aligns with positioning CSE students as critical thinkers and encourages them to complete steps related to tasks as text or content users, but also to evaluate and question things about the process. For example, does it accomplish the intended goals? Is it efficient? Equitable? Viable in multiple environments, etc.?

Identity was a major factor in faculty critical sensemaking of centering student needs in their courses. This aligns with the concern-based adoption model (CBAM), which supports that attending to one's self-concerns is a vital part of the adoption process. Wickerman and Wang [39] explain that faculty take on many identities while sensemaking curriculum changes, establishing identity as the most important element of the process. During the discussion, faculty were struggling to find the connection between their own identity and practices that help students learn in the classroom. According to Wickerman and colleagues [39], during professional development opportunities, faculty take on an

adult learner identity, and if they are unable to make meaningful connections between the proposed change and their previous experience and assumptions, they are resistant to change. Centering the student needs redistributes power within the course and improves opportunities for all people to learn. However, as existing members (faculty) of the discourse community, there may be hesitation about relinquishing control to whoever has access [30]. Though Schildt and colleagues [38] suggest that power is not necessarily restricting access in explicit terms, it may be simply upholding a long-held understanding.

As institutions continue to provide professional development focused on increasing participation, it is vital to not solely focus on how to redesign curriculum and implement pedagogical strategies. Yet, there must be space and opportunity for faculty members to remake their identities regarding their role as instructors and support capacity building for tailoring their course to meet the needs of their students [39]. Power can be both negative/constraining and positive/enabling [38]. Further research is needed to explore how professional developments can be designed and implemented to support faculty identity reconstruction when adopting equitable curriculum redesign and the role of power in that process.

The 4Es heuristic guided faculty to center student identities, experiences, and interests, thus using students' cultural capital as a tool to provide them with access to the discourse community of computer science education. By drawing upon students' existing knowledge and not assuming early or prior exposure to computer science-related concepts, instructors widen the possibilities for participation in the discourse community. Valuing students' opinions and providing opportunities for them to challenge and agree with often taken-for-granted concepts in computer science may also shift their habitus by expanding their thoughts about who gets to be a computer scientist.

4.1. Centering Student Identities

Faculty instructors center student identities most frequently by considering students' interests and experiences as the starting point for engaging in disciplinary practices. Of the seven disciplinary practices connected to engagement, we found that faculty actively centered student identities in several important ways. For example, a discussion about framing the question/problem prompted an instructor to say during an instructional team meeting, "We can't jump right off the bat into how to code—first we need motivation!" The faculty subsequently designed an introductory activity called Rocket Science that they thought would engage their current students based on their interests and agreed on the importance of getting to know the students in each class to tailor future problems and inquiries around their curiosities. The redesign team meeting minutes also provide evidence that faculty were spurred to think about students' prior experiences with coding, opportunities to demonstrate their learning in various modalities, and providing students with space to challenge claims in existing CSE literature and contrast them with their own experiences.

Instructor statements gathered from collaborative planning sessions demonstrated how focusing on eliciting and engineering in the ICS course resulted in conversations among the instructional team of faculty that centered students' previous experiences ("We should also not assume that students are already computer-savvy or have coded before") and guided them to design instructional experiences that scaffold from their existing skills and consider their overall experiences as undergraduate students ("It is important to note that most students in this class are in their first or second semester of college, and could use a little encouragement to build positive study habits in general!"). Instructors also commented on the computer science curriculum: "How would this course change if we truly assumed no prior knowledge? No term is too simple to define those may be the best terms to try to define! The simpler the better!!!" Finally, we found that faculty encouraged student voice in ways not previously performed in the existing curriculum. Faculty designed multiple opportunities for students to comment on their perspectives regarding literature about computer science (various multimodal formats, including the

textbook), to work in small groups to orally present their ideas and questions, and to learn from others.

The intentional practice of beginning from the beginning and not assuming prior knowledge about computer science and computer functioning actively addresses the inequitable representation of students who have early exposure to STEM education or previous experience with computer science. The curriculum redesign represents the ICS faculty's willingness to suspend assumptions about students' knowledge and to explicitly identify, define, and explain both disciplinary content and habits of practice. Faculty also address inequitable representation due to "qualities of difference" by centering students' experiences and feedback. Faculty-centered student experiences by using their knowledge about students' backgrounds to inform the design of their learning activities. Students' interests were used to create contexts for inquiries designed to engage students in computer science practices. The faculty also designed opportunities for students to share their feedback in the form of agreement and challenge as it relates to the multimodal forms of computer science-related content.

4.2. Shifting Power Relations

We sought a redistribution in power relations because students, as novices in the field of computer science, are typically at a disadvantage within CSE. While the faculty are the experts because of their knowledge and experience within the discourse community, it is imperative to acknowledge the value of opening the discourse community to various perspectives that may support and challenge the existing cultural capital and habits of practice. Moje [25] identifies the potential of the 4Es heuristic as a framework in support of achieving social justice in education.

As the faculty considered how to elicit student perspectives and engineered opportunities for them to share their perspectives and feedback, they were simultaneously widening opportunities for participation within the discourse community for students who were previously excluded because of their previous experiences or qualities of difference. Ultimately, drawing upon student experiences and taking nothing for granted as it relates to the content and disciplinary practices of CSE, power was redistributed more in favor of the students than it had been in the original course design. Key changes allowing for shifts in the curriculum toward centering student identity and a redistribution of power in favor of students began with the five big ideas of computer science course content, which illuminated our need for a supplemental framework (4Es heuristic) to attend to supporting student engagement in the discourse community. We identified the 4Es heuristic as a feasible method to support students as they engage with the five big ideas of computer science within their discourse community.

4.3. Limitations and Challenges

Our study reflects data collection and analysis of one ICS course curriculum and two class lectures. We had, as part of our team, two faculty instructors for the ICS course. Also, due to time constraints, we were only able to redesign the curriculum for one course. Thus, we are limited in generalizing the findings of our research to other CSE contexts. In the future, we would recommend recruiting a larger number of faculty to elicit additional responses and feedback on the course redesign. We would also recommend exploring curriculum redesign in multiple contexts before making broad recommendations for CSE.

5. Conclusions

We found the 4Es heuristic to be a useful tool to support the reimagining of an undergraduate ICS course to center student identity and redistribute power in favor of students in CSE. Key changes to the curriculum included identifying the five big ideas of computer science education and designing the pacing guide to encompass instructor guidance for teaching both the disciplinary content and literacy practices required for participation in the discourse community. Using the 4Es framework guided ICS faculty

to consider changes in the curriculum that made the content and disciplinary practices of CSE accessible to all students, regardless of their prior experiences, level of exposure to computer science education, or decision to pursue computer science as a major. The faculty made strides to design activities and opportunities centered on students' experiences, both to help students connect to the content and to honor the value of their individual experiences in support of their learning.

Although the 4Es work in tandem to fully open access to discourse communities, we recognize the significance of the incremental progress that was made when we focused on engagement and elicitation/engineering. Thus, we would recommend beginning with these two areas if one desires to use the 4Es framework to make small changes in their curriculum and instruction. Specifically, the seven disciplinary practices of engagement are: (1) framing questions/problems, (2) working with data, (3) using varied media to consult and produce multiple texts, (4) analyzing, summarizing, and synthesizing data into findings related to the questions posed, (5) examining and evaluating one's claims and the claims of others, (6) communicating claims orally or in writing, and (7) using the cycle of disciplinary practices to provide robust opportunities for educators and CSE faculty to consider centering students and shifting the power dynamics in favor of students in their ICS courses.

Future research could extend this work by using the 4Es heuristic to redesign ICS courses or other courses in CSE. It would also be beneficial to continue working with faculty that are instructors of CSE courses to investigate how to support them through the process of shifting their curricula and pedagogies. Researchers may also consider eliciting pre- and post-student evaluations and feedback on their experiences as students in the CSE courses that are designed using the 4Es heuristic. It may be particularly useful to consider the perspectives of students who are uniquely impacted and marginalized by inequities in CSE.

Author Contributions: Conceptualization, F.D.S., S.M.R. and J.M.; methodology, F.D.S., S.M.R. and B.C.; formal analysis, F.D.S., S.M.R., B.C.; investigation, K.L., J.R. and J.M.; writing—original draft preparation, F.D.S. and S.M.R.; writing—review and editing, F.D.S. and S.M.R.; visualization, F.D.S. and S.M.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Kallia, M.; Cutts, Q. Re-examining inequalities in computer science participation from a Bourdieusian sociological perspective. In Proceedings of the 17th ACM Conference on International Computing Education Research, New York, NY, USA, 16–19 August 2021; pp. 379–392. [CrossRef]
2. Wang, J.; Moghadam, S.H. Diversity barriers in K-12 computer science education: Structural and social. In Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE, Bologna, Italy, 3–5 July 2017; pp. 615–620. [CrossRef]
3. Gretter, S.; Yadav, A.; Sands, P.; Hambrusch, S. Equitable learning environments in K-12 computing: Teachers' views on barriers to diversity. *ACM Trans. Comput. Educ.* **2019**, *19*, 1–16. [CrossRef]
4. Turnbull, S.M.; Locke, K.; Vanholsbeeck, F.; O'Neale, D.R. Bourdieu, networks, and movements: Using the concepts of habitus, field and capital to understand a network analysis of gender differences in undergraduate physics. *PLoS ONE* **2019**, *14*, e0222357. [CrossRef] [PubMed]
5. Bourdieu, P. *Distinction: A Social Critique of the Judgment of Taste*; Routledge: London, UK, 1984.
6. Nash, R. Bourdieu, 'Habitus', and Educational Research: Is It All Worth the Candle? *Br. J. Sociol. Educ.* **1999**, *20*, 175–187. [CrossRef]
7. Metcalf, H.E.; Crenshaw, T.L.; Chambers, E.W.; Heeren, C. Diversity across a decade: A case study on undergraduate computing culture at the University of Illinois. In Proceedings of the 49th ACM Technical Symposium on Computer Science Education, Baltimore, MD, USA, 21–24 February 2018; pp. 610–615.
8. Reeves, S. Why do they drop? *ICER submitted*.
9. Luke, A.; Freebody, P. A map of possible practices: Further notes on the four resources model. *Pract. Prim.* **1999**, *4*, 5–8.

10. Taylor, L.D.; Williams, K.L. Critical Sensemaking: A Framework for Interrogation, Reflection, and Coalition Building toward More Inclusive College Environments. *Educ. Sci.* **2022**, *12*, 877. [CrossRef]
11. Cheryan, S.; Ziegler, S.A.; Montoya, A.K.; Jiang, L. Why are some STEM fields more gender-balanced than others? *Psychol. Bull.* **2017**, *143*, 1–35. [CrossRef] [PubMed]
12. Höhne, E.; Zander, L. Sources of male and female students' belonging uncertainty in the computer sciences. *Front. Psychol.* **2019**, *10*, 1740. [CrossRef] [PubMed]
13. Hernandez, C. Undergraduate Women of Color in Computer Science: How Social and Academic Experiences Shape Sense of Belonging (Publication No. 30312468). Ph.D. Thesis, Rutgers, The State University of New Jersey, New Brunswick, NJ, USA, 2023.
14. Espinosa, L. Pipelines and pathways: Women of color in undergraduate STEM majors and the college experiences that contribute to persistence. *Harv. Educ. Rev.* **2011**, *81*, 209–241. [CrossRef]
15. Ong, M.; Wright, C.; Espinosa, L.; Orfield, G.; Ong, M. Inside the double bind: A synthesis of empirical research on undergraduate and graduate women of color in science, technology, engineering, and mathematics. *Harv. Educ. Rev.* **2011**, *81*, 172–208. [CrossRef]
16. Shein, E. Broadening the path for women in STEM. *Commun. ACM* **2018**, *61*, 19–21. [CrossRef]
17. Fitzsimmons, S. A Comparative Study of Male and Female Undergraduate Computerscience Students' Educational Pathways (Publication No. 28322950). Ph.D. Thesis, University of South Florida, Tampa, FL, USA, 2021.
18. Lavakumar, A. Fostering Computer Science Identity among Underrepresented Minority Students: An Exploration of Contributing Factors (Publication No. 28644435). Ph.D. Thesis; Regis College: Weston, MA, USA, 2022.
19. Shanahan, T.; Shanahan, C. What Is Disciplinary Literacy and Why Does It Matter? *Top. Lang. Disord.* **2012**, *32*, 7–18. [CrossRef]
20. Hayden, H.E.; Eades-Baird, M. Disciplinary literacy and the 4Es: Rigorous and substantive responses to interdisciplinary standards. *Lit. Res. Theory Method Pract.* **2020**, *69*, 339–357. [CrossRef]
21. Moje, E.B. Foregrounding the Disciplines in Secondary Literacy Teaching and Learning: A Call for Change. *J. Adolesc. Adult Lit.* **2008**, *52*, 96–107. [CrossRef]
22. Shanahan, T.; Shanahan, C. Teaching disciplinary literacy to adolescents: Rethinking content-area literacy. *Harv. Educ. Rev.* **2008**, *78*, 40–59. [CrossRef]
23. Shanahan, C.; Shanahan, T.; Misischia, C. Analysis of expert readers in three disciplines: History, mathematics, and chemistry. *J. Lit. Res.* **2011**, *43*, 393–429. [CrossRef]
24. Jackson, M. The Effects of a Disciplinary Literacy Framework on Scientific Literacy and Student Engagement in the Middle School Science Classroom. Ph.D. Thesis, University of Massachusetts Lowell, Lowell, MA, USA, 2022.
25. Moje, E.B. Doing and teaching disciplinary literacy with adolescent learners: A social and cultural enterprise. *Harv. Educ. Rev.* **2015**, *85*, 254–278. [CrossRef]
26. Akkuş Çakır, N.; Gass, A.; Foster, A.; Lee, F.J. Development of a game-design workshop to promote young girls' interest towards computing through identity exploration. *Comput. Educ.* **2017**, *108*, 115–130. [CrossRef]
27. Main, J.B.; Schimpf, C. The underrepresentation of women in computing fields: A synthesis of literature using a life course perspective. *IEEE Trans. Educ.* **2017**, *60*, 296–304. [CrossRef]
28. Clear, T. Critical enquiry in computer science education. In *Computer Science Education Research*; Taylor & Francis: New York, NY, USA, 2004; pp. 101–125.
29. Hubbard Cheuoua, A. Confronting inequities in computer science education: A case or critical theory. In Proceedings of the 52nd ACM Technical Symposium on Computer Science Education, Virtual, 13–20 March 2021; pp. 425–430.
30. Moje, E.B.; Lewis, C. Examining opportunities to learn literacy: The role of critical Sociocultural literacy research. In *Reframing Sociocultural Research on Literacy*; Lewis, C., Enciso, P., Moje, E.B., Eds.; Routledge: London, UK, 2007; pp. 15–48.
31. Ben-David Kolikant, Y. (Some) grand challenges of computer science education in the digital age: A socio-cultural perspective. In Proceedings of the 7th Workshop in Primary and Secondary Computing Education, Hamburg, Germany, 8–9 November 2012; pp. 86–89.
32. Vrieler, T.; Salminen-Karlsson, M. A sociocultural perspective on computer science capi tal and its pedagogical implications in computer science education. *ACM Trans. Comput. Educ.* **2022**, *22*, 1–23. [CrossRef]
33. Freire, P. Pedagogy of the oppressed. In *Toward a Sociology of Education*; Routledge: London, UK, 2020; pp. 374–386.
34. Luke, A.; Freebody, P. Shaping the social practices of reading. In *Constructing Critical Literacies: Teaching and Learning Textual Practices*; Muspratt, S., Luke, A., Freebody, P., Eds.; Hampton: Cresskill, NJ, USA, 1997; pp. 185–225.
35. Firkins, A.S. The Four Resources Model: A useful framework for Second Language teaching in a Military Context. *Technol. Stud. Inst. J.* **2015**, *2015*, 1–5.
36. Freebody, P.; Luke, A. 'Literacies' Programs: Debates and Demands in Cultural Context. *Prospect* **1990**, *5*, 85–94.
37. Weick, K.E. *Sensemaking in Organizations*; SAGE: Thousand Oaks, CA, USA, 1995.
38. Schildt, H.; Mantere, S.; Cornelissen, J. Power in Sensemaking Processes. *Organ. Stud.* **2020**, *41*, 241–265. [CrossRef]
39. Wickersham, K.; Wang, X. Making sense to make change: How community college faculty percep tions of math contextual-ization shape instructional change. *Community Coll. Rev.* **2022**, *50*, 3–29. [CrossRef]
40. Coulter, D.-M. Adjunct Faculty Sensemaking in the Context of a Student Success Initiative at a Community College Initiative at a Community College. Ph.D. Thesis, Rowan University, Glassboro, NJ, USA, 2016.
41. Caro, V.; Carter, B.A.; Millunchick, J.; Reeves, S. Teaching crystal structures in undergraduate courses: A systematic review from a disciplinary literacy perspective. *Chem. Educ. Res. Pract.* **2023**, *24*, 394–406. [CrossRef]

42. Stevens, S.; Sutherland, L.; Krajcik, J. *The Big Ideas of Nanoscale Science and Engineering: A Guidebook for Secondary Teachers*; NSTA Press: Richmond, VA, USA, 2009.
43. Tan, Y.; Ruis, A.R.; Marquart, C.; Cai, Z.; Knowles, M.A.; Shaffer, D.W. Ordered network analysis. In *Advances in Quantitative Ethnography*; Damşa, C., Barany, A., Eds.; ICQE 2022; Communications in Computer and Information Science; Springer: Cham, Switzerland, 2023; Volume 1785. [CrossRef]
44. Marquart, C.L.; Hinojosa, C.; Swiecki, Z.; Eagan, B.; Shaffer, D.W. Epistemic Network Analysis (Version 1.7.0) [Software]. 2021. Available online: http://app.epistemicnetwork.org (accessed on 2 January 2024).
45. Bowman, D.; Swiecki, Z.; Cai, Z.; Wang, Y.; Eagan, B.; Linderoth, J.; Williamson Shaffer, D. The math ematical foundations of epistemic network analysis. In *Proceedings of the Advances in Quantitative Ethnography: Second International Conference, ICQE 2020, Malibu, CA, USA, 1–3 February 2021*; Ruis, A., Lee, S.B., Eds.; Springer: Berlin/Heidelberg, Germany, 2021; pp. 91–105.
46. Shaffer, D.W.; Collier, W.; Ruis, A.R. A tutorial on epistemic network analysis: Analyzing the structure of connections in cognitive, social, and interaction data. *J. Learn. Anal.* **2016**, *3*, 9–45. [CrossRef]
47. Larsen, E.A.; Stubbs, M.L. Increasing diversity in computer science: Acknowledging, yet moving beyond, gender. *J. Women Minor. Sci. Eng.* **2005**, *11*, 139–170. [CrossRef]