Achieving Gender Balance through Creative Expression

William H. Bares, Bill Manaris, Renée McCauley, Christine Moore

Computer Science
College of Charleston
Charleston SC USA

{BaresWH | ManarisB | McCauleyR | MooreC}@cofc.edu

ABSTRACT

Increasing gender balance in computing is widely recognized by academic institutions, industry, and government agencies as an imperative. This paper describes how providing opportunities for creative expression early on and throughout the four years of an undergraduate computing degree achieves this goal. Prior studies have demonstrated that opportunity for creative expression has a positive effect on recruitment and retention of women, while conversely, lack of opportunity for creative expression has a negative effect. We describe our approach, integrated into a four-course sequence combining computing and the arts, so that other institutions may consider adopting it. Results of a six-year longitudinal study show that these courses attract, retain, and graduate 46% female students, while the corresponding number in our ABET-accredited Computer Science curriculum is approximately 20%. The paper concludes with general observations and suggests directions for future action in CS curricula design.

CCS CONCEPTS

• Computing Milieux • Computers and Education • Computer and Information Science Education

KEYWORDS

Creativity, art, contextualized computing, broadening participation, curriculum

ACM Reference format:

William Bares, Bill Manaris, Renée McCauley, and Christine Moore. 2019. Achieving Gender Balance through Creative Expression. In *Proceedings of ACM Special Interest Group on Computer Science Education (SIGCSE'19)* February 27-March 2, 2019, Minneapolis, MN, USA, ACM, NY, NY, USA, 7 pages. <https://doi.org/10.1145/1234567890>

1 Introduction

“Why, with so much sustained effort by so many individuals and organizations, is progress toward gender equity so slow?”, Tims wonders in [32]. In particular, the national graduation rate of

female students in computer science is close to 20% (i.e., 17.9% according to the Taulbee Survey [36] or 22.1% according to the ACM-NDC Survey [32]). This paper describes our experiences, over a six-year period, to achieve gender balance in computing.

The literature is full of articles offering perspectives and potential remedies to this gender balance problem. Some of these articles show that integrating creative expression with computer science results in courses that are often gender-balanced. For instance, Georgia Tech’s bachelor’s degree in Computational Media is reported to have 45.3% female enrollment, compared to 17.91% in the computer science degree [19]. Similarly, Barker reports that the Technology, Arts, & Media degree at the University of Colorado, Boulder has a 44.8% female enrollment [2, 3].

1.1 Our Experience – Computing in the Arts

At our institution, a six-year longitudinal study shows that our Computing in the Arts (CITA) degree attracts, retains, and graduates 46% female students, compared to approximately 20% in our ABET-accredited computer science (CSCI) degree. These figures were collected by analyzing anonymized course enrollment and graduation data supplied by our Office of Institutional Research. See [1] for further details of the study methodology and limitations. At the core of this CITA degree is a sequence of semester-long creative experiences that synthesize computing and one or more arts areas (music, visual art, theatre, or digital media). These synthesis courses are designed to ensure that students learn to use computational techniques, in general, and computer programming, in particular. Learning computer programming is central. It is used as a medium to create avant-garde artistic experiences, as well as to develop innovative computer applications informed / guided by artistic theory and practice.

More precisely, Table 1 shows enrollments by gender in CITA synthesis courses, over a six-year period, against comparable CSCI courses. These are comparable pairs of required courses, taken by respective majors (CITA and CSCI), during the freshman, sophomore, junior, and senior years. As can be seen, CITA courses consistently attract a higher percentage of female students than the corresponding CSCI courses. It should be noted that, while these CITA courses are required for CITA majors, the first- and second-year CITA courses are also frequently taken by non-majors. CITA majors also take CS1 and CS2, so they impact the higher percentages of female students in those courses as well. We also found that 44% of non-computing students who choose

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

SIGCSE '19, February 27-March 2, 2019, Minneapolis, MN, USA.

© 2019 Association for Computing Machinery.

ACM ISBN 978-1-4503-5890-3/19/02...\$15.00

<https://doi.org/10.1145/1234567890>

to enroll in a first-year CITA synthesis course are Arts or Business majors, while 49% of non-computing students attracted to CS1 are Lab Sciences and Math majors. Based on the first course alone, CITA appears to attract a different population from that of our Computer Science degree program [1].

Table 1. Enrollments in synthesis courses (2011-2017)

CITA Synthesis Courses				CSCI Courses			
Course	F	M	F%	Course	F	M	F%
Synth1	224	384	36.8	CS1	350	860	28.9
Synth2	50	107	31.9	CS2	217	650	25.0
Seminar	24	37	39.4	Soft Arch	56	241	18.9
Capstone	21	27	43.8	Practicum	45	199	18.4

This paper complements and completes the reporting in [1] by providing detailed information on the synthesis courses, so that other institutions may explore adopting this approach. This paper is organized as follows. Section 2 explores foundational, creative computing courses which influenced the design of the CITA synthesis courses. Section 3 describes the contents of the courses in detail for potential replication. Section 4 discusses how other institutions may apply similar approaches (and possibly degrees). Finally, Section 5 adds concluding remarks.

2 Foundations

2.1 Contextualized Computing

Contextualized computing focuses on applying computation in a particular domain of interest. Accordingly, contextualized computing pedagogy focuses on providing well-designed educational experiences that teach computing to develop applications in a particular context (e.g., Physics, or Media). Guzdial [17] describes the rationale for contextualized computing as follows:

A context provides relevance and improves retention. Students are more likely to stick around if they understand the value of what they're learning. There is no evidence that they learn the material better, but they're more likely to be around in the second class, even if that's a normal decontextualized course. [p. 6]

Contextualized computing courses have been designed to overcome the perceptions among students that computer science is too abstract, emphasizes tedious details, and offers limited opportunities for creative expression. Cooper and Cunningham describe different efforts under the umbrella of contextualized computing, which aim to engage and retain students in learning computer science [10]. Uludag et al. [35] explore the combination of complementary technologies to increase appeal to broad audiences and attract both non-majors and majors. They incorporate visual programming with Scratch, App Inventor for mobile phones, and LEGO Mindstorms for robotics and engineering. La Marche [22] presents a contextualized-computing course centered on user-interface design. He discusses how

contextual pedagogy may “lower failure rates, improve retention, increase student self-efficacy regarding programming, and increase positive views about computer science.”

A major contribution in the area is by Guzdial [16], who along with various collaborators [14, 25, 28] introduced the concept of Media Computation [18]. This led to a Media Computation “movement” consisting of a large community of educators adopting Guzdial’s pedagogical materials and introducing computer programming in the context of the arts. This resulted in our first contextualized-computing course offered in Fall 2006. Our course combined Python, fractals, L-Systems, and jMusic (a Java-based system for composing music with computers). Initially called “Media Computation”, we soon renamed it “Computers, Music, and Art” (Fall 2007). Eventually, this became one of our required introductory courses in our computing in the arts degree (further discussed in section 3).

Overall, media computing courses have demonstrated lower failure rates compared to traditional computer science courses [12, 26, 29]. Media computation courses have induced greater interest in students to take additional media computation courses [13, 29, 30]. Guzdial reports results of a 10-year study of a media computation course showing a drop-rate of 12% (compared to 50% for conventional CS1). This course achieved a 40% enrollment of female students, with no notable differences in performance by gender. In a multi-year study, a combination of courses including media computation demonstrated an 18% improvement in retention one year later. This eventually led to development of the Georgia Tech Bachelor’s degree in Computational Media, which in 2014 was reported to have 45.32% female enrollment as compared to only 17.91% female enrollment in the computer science major [19].

2.2 Creativity and Broadening Participation

Even though the number of female students attending college exceeds the number of male students, female students are less likely to choose science and technology careers [28]. The lack of women in the sciences has been likened to a “leaky pipeline” [7] from which female students leave at a greater rate than their male counterparts. CS enrollments are at an all-time high [9] at all types of institutions, (public and private, doctoral-granting and non-doctoral granting) with doctoral-granting reporting the largest increases. But women continue to be underrepresented, as the most recent Taulbee survey shows undergraduate CS enrollments by gender are 81.7% male to 18.35 female.

...the low number of women in CS hurts the ability of businesses to hire qualified employees. Another consideration is that the field loses an opportunity to harness women's perspectives and innovativeness. Important breakthroughs, creativity, and true innovation require outside-of-the-box thinking, which may be fostered by diversity [6].

Over the last few decades many have attempted to find reasons for this underrepresentation. Blickenstaff analyzes this issue and notes a range of possible causes identified in the literature,

including differences in academic preparation for success in sciences, lack of female role models, and a negative perception of the sciences [7]. Others have identified issues related to lower levels of confidence [5], misperceptions of ability as compared to men [11], and self-efficacy [5, 11]. Others have noted that when women do start the degree, they drop out in greater numbers than their male counterparts [8, 9, 21].

Plucker et al. 2004 [24] define creativity as “the interaction among aptitude, process, and environment by which an individual or group produces a perceptible product that is both novel and useful as defined within a social context.” Expressions of creativity are individualized “novel and personally meaningful interpretation of experiences, actions, and events.” [4]. Research into the benefits of contextualized computing shows that creative expression is a significant factor for attracting and retaining students [13, 29, 30]. On the other hand, the lack of opportunity for creative expression is a significant factor as to why students, and especially women, leave Computer Science [10, 12]. Programming classes that afford students opportunities to be creative reported higher levels of motivation and interest [23]. Studies of software engineering practice have found that developers prefer to work on activities that they perceive as being creative [15]. Consequently, it seems advantageous to consider offering courses that integrate creative expression and computer science.

3 Synthesis Courses for Creative Expression

The synthesis courses, which contextualize art and computing, are vertically integrated throughout our CITA curriculum, and thus provide opportunities for creative expression throughout the undergraduate experience. However, the first- and second year courses are also taken by non-CITA majors, so even at institutions without dedicated degrees, courses integrating creative expression with computer science could bring more women to computing. Figure 1 illustrates the pathway through the CITA core curriculum.

Figure 1: CITA curriculum pre-requisite course flow chart.

In addition to what is shown in Figure 1, CITA majors complete 9 credits in upper-division CSCI coursework, and 18

credits in their art concentration area. The first-year synthesis (Synthesis 1) courses seek to introduce basic programming concepts situated in an accessible and creative arts-themed context. The second-year synthesis course engages students in applying software development practices including iterative design, prototyping, peer evaluation of prototypes, and testing in the context of game programming. In the third-year synthesis course (Seminar), students develop a proposal for a capstone project, which synthesizes creativity in the arts with the tools and conceptual modeling systems of computing. The prerequisites for Seminar also include 3 courses in the student’s chosen arts concentration (e.g., music, visual art, theatre, etc.).

3.1 Synthesis 1: Intro to Computing in the Arts

The first-year synthesis courses allow students to choose between creating animated short 3-D films, musical compositions and 2-D visualizations. These courses seek to introduce basic programming concepts situated in an accessible and creative arts-themed context. These courses do not expect students to have prior programming experience; they have no prerequisites and so are taken by students from various majors. However, these courses are required for CITA majors, and are recommended for Computer Science majors who have no programming experience, prior to enrolling in the first Computer Science course. These courses could be described as CS0.5 courses.

The first year-synthesis courses share the following computing learning outcomes:

- apply numeric and string data types to represent information,
- use variables in program development,
- understand arithmetic operators and use them to design expressions,
- understand for-loops and use them to design processes involving repetition,
- understand if statements and use them to design processes involving selection,
- understand functions and use them to design processes involving modularization,
- use predefined classes in program development (object-based programming),
- learn basic principles for group collaboration.

These computing learning outcomes span the AP Computer Science Principles Curriculum. This includes computational thinking practices (connecting computing, creating computational artifacts, abstracting, analyzing problems and artifacts, communicating, and collaborating), as well as the 7 Big Ideas: creativity, abstraction, data and information, algorithms, programming, the Internet, and global impact.

3.1.1 Animation and Virtual Worlds

The arts-related learning outcomes of this course highlight parallels between the software design and artistic design processes. Notably, both processes entail repeated cycles of design, construction, and evaluation, in which an artefact

becomes gradually more refined. In the themed context of computer animation, students see the parallels between hand-drawn storyboards and algorithms written in pseudocode. All course assignments encourage creativity by defining a fixed set of requirements or demonstrable artistic or technical features leaving the students free to create their own characters, objects, scenes, or stories. The course currently uses Autodesk Maya with its built-in interactive Python code window, but could also use Blender 3-D. Example assignments include:

Modeling in 3-D: Create and texture paint an original 3-D character. First draw front and side profile sketches, which are imported into a 3-D modeling application to serve as guides. Apply polygon modeling tools to extrude, translate, rotate, scale, and subdivide to form desired the polygon mesh shape. Assign UV-texture coordinates to the model to create unwrapped 2-D outlines of patches of the model's skin. Use a 2-D paint program to color this unwrapped 2-D texture image. Submit the 3-D model, texture image, and rendered images showing the character from front and side views to compare the finished model to the initial design sketches. Students are free to model their work after an existing favorite character or original design.

Scripted scene layout: Create a 3-D scene composed of multiple instances of character models and objects. Students apply concepts of loops and selection statements to create instances (or copies) of their 3-D models, whose position and size are set by invoking the math library random number function. A selection statement inside the loop is used to randomly choose one of several 3-D models to be instanced. Students may use their 3-D character model created in the modeling assignment or borrow characters created by classmates to create a more complex scene. For example, students may create a space scene comprised of multiple different types of space craft and asteroids randomly scattered in space.

Scripted animated scene: Create a simulated particle system effect by applying concepts of loops, lists, and objects. Each particle is an object that has attributes of position and velocity. Students use a count-controlled loop to initialize the starting position and velocity of each particle. They next apply a count-controlled loop to simulate the animation for each particle. Nested inside the count-control loop is a conditional loop that animates the current particle until a stopping condition is reached, e.g. rain drop particle reaches the ground.

Final project: The course concludes with a two-person team project to create an original short animated story in which the team demonstrates creative contributions in story, characters, props, music, sound, or animation. Each student must individually demonstrate creative contribution in any two of these listed aspects plus original code to script scene layout or animation. Students practice their presentation skills by introducing their concept, presenting storyboards, blocking diagrams, and highlighting their respective contributions before screening their animated short film.

3.1.2 Computers, Music, and Art

This course's topics include computer programming and music through the study, transcription, and creation of musical works.

Music topics may include notation, scales, key signatures, intervals, chord construction, sight-singing, ear training, and readings in music history and aesthetics. Computing topics include data types, variables, assignment, selection, iteration, lists, functions, classes, events, and graphical user interfaces. Students are introduced to the computer as a musical instrument and as a creative environment to develop fluency with musical practices such as algorithmic composition, developing simple computer instruments, and composing exercises based on models in electroacoustic music and minimalism. Students are required to produce various musical artifacts, ranging from the transcription of musical material (e.g., a favorite song or piece) to the development of solo and interactive group performances.

The course is taught in Python, using an approach blending programming and creative activities. It utilizes a special programming environment for creative programming in music and art, called JythonMusic [20]. JythonMusic provides libraries for music making, image manipulation, building graphical user interfaces (GUIs), and connecting computers to external MIDI and open sound control (OSC) devices, such as digital pianos, smartphones and tablets. Example assignments include:

Transcribe a favorite piece of music: Create a Python program that generates an interesting piece of music. This assignment focuses on creating interesting music with Python, MIDI melodies and harmony, and MIDI percussion. To prepare for this assignment, we study a program that creates a drum pattern consisting of kick, snare and hi-hat sounds, a program that creates a J.S. Bach canon, and a program that creates a minimalist piece ("Piano Phase" by Steve Reich). Students are presented with two options: either (a) find an interesting minimalist piece (e.g., by Philip Glass) and transcribe it, or (b) create your own song in JythonMusic. If a student chooses option (b), then the song needs to have both a melody and drums (all in MIDI).

Sonify an interesting image: Select an interesting image, and sonify (create music from it) using JythonMusic. Design the musical parameters (pitch, dynamics, timbre, panning, etc.) to use. You could use only part of the image, or different parts of the image for different part of the music. For example, convert a single ("rhythmically" interesting) row (or column) to a bass line. Or, create a recurring theme by sonifying a small (interesting) area of the image, and repeating it. Decide ahead of time if you will create continuous, chromatic, or diatonic music. Explore high-level structure of your music. Finally, you may add up to 20% of supporting (non-image generated) musical material, e.g., a low-pitched drone or an ambient background. Examples of student submissions may be found at <http://bit.ly/visualSoundscapes2>.

Final project: One way to conclude the course is with a laptop orchestra performance. Laptop orchestras became popular after the Princeton Laptop Orchestra (PLOrk) [33]. Such assignments make computing tasks more communal, which is appealing to students, especially women. Students program their computer in real time to perform musical tasks together with other members. For instance, in Fall 2010, we assigned Terry Riley's "In C", in lieu of a comprehensive final exam [27]. Students pre-coded the 53 different patterns of the piece in one or more timbres (128 MIDI

instruments). During the performance, students wrote necessary code to generate these different patterns in real time.

3.1.3 Digital Media Programming

Students learn the fundamentals of computer programming within the context of digital media and artistic expression. In addition to the computer science outcomes outlined above, the course also features design-related objectives. The application of foundational principles of design is the key to creating effective designs that have both visual and communicative success. The course requires students to apply the concept development process, which encompasses a sequence of steps that is common among problem-solving methodologies.

The course is taught primarily using Processing, a language for programming images, animation, sound, and games. Employing Java libraries as a base, Processing allows students to create compelling and beautiful designs while learning the fundamentals of computer programming. As illustrated in the example assignments discussed below, the course begins with creating static sketches in order to teach students how to code shapes, colors, and more complex composites on-screen. Then, as students increase their programming and design skills, programs become event-driven and interactive.

Let's Get Animated: Students create an aesthetically pleasing design using primitive geometric shapes. Then, they make it interactive. At this point in the course, students have learned how to code shapes, use built-in functions, such as `random()`, and create variables and control structures.

Create Visual Patterns: Students use some form of iteration to create a visually interesting, static pattern. Additionally, they use Boolean variables to overlay an additional pattern when the user interacts with the design.

Midterm Project: Halfway through the semester, students use functions or objects to create a more intricate animated design. They create at least three custom functions, and practice incremental development / testing to ensure that each function works correctly as a separate unit of code.

Final Project: Students design an interactive art composition or a game, while demonstrating a composite of knowledge and skills gained during the semester. At minimum, students are required to use classes and other programming constructs such as loops, conditionals, strings, and arrays. The project is judged on the effective implementation of design principles in the code. For example, create a three-level game in which a given number of oranges must be caught with the bucket before hitting the ground.

3.2 Synthesis 2: Game Programming

The second-year synthesis course engages students to create two-dimensional games from their favorite genres, such as arcade, role-playing, fantasy, sports, serious game, etc. This course teaches students to apply an iterative software design process involving repeated cycles of prototyping and evaluation to build interactive games. Students learn essential skills in Computer Science by developing and testing algorithms. The course spends a comparable amount of time developing aesthetic and design skills.

The computing learning outcomes include creating, testing, and refining prototypes (paper, digital, and code), designing and applying test cases, applying classes to represent game objects, applying lists of objects to animate objects and check for collisions, applying two-dimensional lists to represent tile-based maps, and applying finite-state machines to control behavior of game objects, and maze navigation or wandering algorithms. For example, students apply finite-state machines to represent the chase and evade behaviors of ghosts in Pac-Man. Students also research and discuss profiles of diverse game designers and how serious games are used to communicate messages to audiences around the world.

The design outcomes are adapted from *Game Design Workshop* by Tracey Fullerton. Design outcomes include crafting aspects of a game to achieve desired player experience goals, creating engaging characters and stories, creating appealing interfaces that offer feedback, and developing balanced play mechanics that put the player in a state of “flow,” where the level of challenge adapts to match a player’s skill. Assignments define fixed artistic and technical requirements freeing students to define their own game objectives and artwork design. Typical assignments include:

Serious arcade game: Design a serious game based on the classic Activision Kaboom game mechanic, in which the player’s character must move to catch falling objects. The game should raise awareness of a cause for social good. Develop your design through a paper prototype, feedback from testing your paper prototype with classmates, design test cases for collision interaction and application of game rules, code, and evaluate the resulting game using your test cases. Students take turns playtesting completed games in class and providing written feedback to the developer.

Final project: Design, implement, and test a maze-based game that demonstrates use of multiple classes to represent game objects and the tile maze. Apply finite-state machines to direct the behavior of non-player game objects. Apply collision detection to prevent the player and autonomous non-player objects from passing through barriers. Students are free to design game rules, art, and sounds to express their creativity. Develop design documentation that describes the objective, rules of play, maze, and game objects. Write a detailed testing plan and player-tester checklist that will be used to evaluate program correctness. Testing plan checklist items must be self-contained objective statements of expected correct behaviors. For example, the player-controlled object will re-appear on the opposite side of the screen when it goes beyond one of the four screen edges. Students give presentations of their game designs and take turns playtesting each other’s games and completing playtest checklists.

This course could be taught using any number of development environments, but we have settled on Processing, since it bridges the Python and Java languages used in our CS1 and CS2 courses, both required for CITA majors. Again, CS1 is a prerequisite for this course. Processing also exposes students to a wide-variety of audio, visual, and media toolkits, which many students choose to use in the third-year and fourth-year synthesis projects.

3.3 Synthesis 3: Seminar

The third-year seminar and fourth year-capstone courses offer an even greater range of creative expression and exposure to software engineering. Students are encouraged to explore, design, and share their original concepts in a supportive, cooperative, and active-learning environment. The seminar and capstone experiences take place over a two-semester course sequence.

In the seminar students conduct literature searches to identify prior works that blend art and computing. They then brainstorm and refine their own artistic project ideas, through the same process of iterative prototyping practiced in the second-year synthesis course. In the most recent offering of the course, students must interact with faculty mentors in art and computing, who offer early feedback on artistic and computing aspects of the students' project ideas. These early contacts help students to (a) create projects that better synthesize art and computing, (b) identify and correct problems early on, and (c) help students to change directions when necessary.

By the end of the seminar, students must produce a working code prototype, possibly using and extending an existing code base for their chosen programming language / environment (e.g., Processing) and technology (e.g., Kinect sensors). Students independently explore and select development tools, such as Arduino, Processing, Java, and Unity Game Engine.

The seminar culminates with each student producing a detailed project proposal, which describes how the project integrates art and computing, illustrates its expected outputs and behaviors using mock displays or audio samples, details art and coding tasks, and estimates completion times for each task.

3.4 Synthesis 4: Capstone

During the fourth-year capstone practicum, students implement their projects according to the task descriptions and timeline milestones defined in their proposals. Course activities include:

- demonstrating progress towards timeline milestones;
- presenting work-in-progress demonstrations to the class;
- consulting with one art mentor and one computing mentor to solicit feedback on progress;
- blogging or journaling regularly to detail progress;
- offering constructive feedback to third-year synthesis students as they work through the process of identifying their projects;
- making a formal presentation of the completed project (as final exam); and
- submitting written report summarizing technical details of the solution and reflecting upon the completed work.

For example, one recent capstone student developed an interactive portfolio of autonomous virtual 3-D creature sculptures. The student modeled original 3-D creatures and brought them to life via C# programming in Unity Game Engine. The autonomous creatures respond to one another's presence, thus exhibiting variations in flocking, fleeing, and pursuit behaviors. This typical project exemplifies how creative expression is applied in both the arts and technology within the

CITA degree. In addition, as the students graduate and seek employment or placement in graduate school, these projects become an invaluable part of their application portfolio.

4 Lessons to Apply

Similarly to us, independent CS education researchers report that synthesizing computer science with creative expression improves gender balance. Even if one cannot implement a whole curriculum, starting small – even at the level of a single course – has a positive impact. For instance, in our case, the courses “Computers, Music, and Arts” and “Game Programming” were offered before our CITA degree was designed (Fall 2006). We observed that these courses attracted many non-majors and women, encouraging us to create the CITA degree. Introducing opportunities for creative expression anywhere within a traditional Computer Science curriculum could help. For example, the concept of lists of objects may be taught in a typical CS2 or Data Structures course by implementing insert, search, and delete operations for student records. To offer greater opportunity for creative expression, lists of objects could be used to create an animated particle system effect. Other examples are described in [24]. Moreover, any computing (computer science, information systems, etc.) capstone sequence could involve development of art-inspired projects, offering creative experiences for students.

5 Conclusion

This paper presented our experiences with curriculum design that supports gender balance in computing. A six-year longitudinal study demonstrates that combining creative expression and the arts with core computer science courses (including computer programming, data structures and software engineering) results in graduating 46% female students (compared to the 20% nationwide statistic). This is a significant finding. Moreover, 72% of CITA graduates are placed in CITA-related jobs or graduate studies that involve computing or arts, with a majority of them in jobs that require their computing skills. This indicates that a CITA-like degree is a viable complement to a traditional Computer Science degree, as it prepares students for careers in computing, while at the same time equalizing the gender disparity found in traditional computing degrees.

In closing, by expanding our computing curricula to encompass the more creative aspects of human endeavor and interests, we may attract female students in approximately equal numbers to those of their male counterparts. This is a significant realization for designing future, gender-balanced computing curricula.

6 Acknowledgements

This work has been funded in part by NSF DUE (1323605 and 1044861).

REFERENCES

- [1] Bares, W.H., Manaris, B. & McCauley, R. Gender Equity in Computer Science through Computing in the Arts – A Six-Year Longitudinal Study, *Computer Science Education*, 28, 3 (Sep. 2018), 191-210.

- [2] Barker, L. J., Garvin-Doxas, K., & Roberts, E. What can computer science learn from a fine arts approach to teaching? In *Proceeding of the 36th ACM Technical Symposium on Computer Science Education* (St. Louis, MO, USA - February 23 - 27, 2005), ACM, NY, 421-425.
- [3] F Barker, L. J. email communication, February 28, 2018.
- [4] Beghetto, R. A. & Kaufman, J. C. Toward a broader conception of creativity: A case for mini-c creativity. *Psychology of Aesthetics, Creativity, and the Arts*, 1, 2 (May 2007), 73–79.
- [5] Beyer, S. Gender differences in the accuracy of self-evaluations of performance. *Journal of Personality and Social Psychology*, 59, 5 (Nov. 1990), 960-970.
- [6] Beyer, S. Why are women underrepresented in Computer Science? Gender differences in stereotypes, self-efficacy, values, and interests and predictors of future CS course-taking and grades. *Computer Science Education*, 24, 2-3 (Sep. 2014), 153-192.
- [7] Blickenstaff, J.C. Women and science careers: leaky pipeline or gender filter?, *Gender and Education*, 17, 4 (Aug. 2006), 369–386.
- [8] Brainard, S. and Carlin, L. 1998. A Six-Year Longitudinal Study of Undergraduate Women in Engineering and Science, *Journal of Engineering Education*, (87)4, pp. 369-375.
- [9] Computing Research Association (2017). Generation CS: Computer Science Undergraduate Enrollments Surge Since 2006. <http://cra.org/data/Generation-CS/>
- [10] Cooper, S. and Cunningham, S. 2010. Teaching computer science in context. *ACM Inroads* 1, 1 (March 2010), 5-8. DOI=<http://dx.doi.org/10.1145/1721933.1721934>
- [11] Else-Quest, N., Mineo, C. and Higgins, A. 2013. Math and Science Attitudes and Achievement at the Intersection of Gender and Ethnicity, *Psychology of Women Quarterly*, (37)3, pp. 293-309.
- [12] Forte, A. and Guzdial, M. Computers for Communication, Not Calculation: Media As a Motivation and Context for Learning, in *Proceedings of the 37th Annual Hawaii International Conference on System Sciences - Track 4 - Volume 4*, (Big Island HI, USA - January 5-8, 2004) IEEE, Washington, DC, USA.
- [13] Forte, A. and Guzdial, M. Motivation and nonmajors in computer science: identifying discrete audiences for introductory courses. *IEEE Transactions on Education*, 48, 2 (May 2005), 248-253.
- [14] Garcia, D. and Barr, V. Guzdial, M. and Malan, D. 2013. Rediscovering the Passion, Beauty, Joy, and Awe: Making Computing Fun Again, Part 6, in *Proceedings of the 44th ACM Technical Symposium on Computer Science Education* (Denver, CO, USA - March 6-9, 2013) ACM, NY, USA, 379-380.
- [15] Gu, M. and Tong, X. Towards Hypotheses on Creativity in Software Development. In *5th International Conference on Product Focused Software Process Improvement (PROFES)*, (Kausai Science City, Japan - June 13-15, 2005) Springer, NY, USA, 47–61.
- [16] Guzdial, M. 2003. A Media Computation Course for Non-majors, *SIGCSE Bulletin*, 35, 3 (Sep. 2003), 104-108.
- [17] Guzdial, Mark. Does Contextualized Computing Education Help?, *ACM Inroads*, 1,4 (Dec. 2010), 4-6.
- [18] Guzdial, M. 2013. Exploring Hypotheses About Media Computation, in *Proceedings of the 9th Annual International ACM Conference on International Computing Education Research*, (La Jolla, CA, USA - August 12-14, 2013) ACM, NY, USA, 19-26.
- [19] Guzdial, M. 2014. The most gender-balanced computing program in the USA: computational media at Georgia Tech, *Computer Education Blog*, September 2, 2014. <https://computinged.wordpress.com/2014/09/02/the-most-gender-balanced-computing-program-in-the-usa/>
- [20] JythonMusic. <http://jythonmusic.org>
- [21] Katz, S., Allbritton, D., Aronis, J., Wilson, C. and Soffa, M. Gender, Achievement, and Persistence in an Undergraduate Computer Science Program, *SIGMIS Database*, 37, 4 (Fall 2006), 42-57.
- [22] La Marche, Jeffrey Steven. User Interface Computation As a Contextualized Approach for Introductory Computing Instruction, in *Proceedings of the 9th Annual ACM International Computing Education Research Conference* (La Holla, CA, USA - August 12-14, 2013) ACM, NY, USA, 179-180.
- [23] Lewandowski, G., Johnson, E. and Goldweber, M. 2005. Fostering a creative interest in computer science. In *Proceedings of the 36th SIGCSE technical symposium on Computer science education* (St. Louis, MO, USA - February 23-27, 2005) ACM, NY, USA, 535-539.
- [24] McCauley, R., Manaris, B., Heise, D., Sheller, C., Jolley, J., and Zaring, A. Computing in the Arts: Curricular Innovations and Results. In *Proceedings of the 48th ACM Technical Symposium on Computer Science Education* (Seattle, WA - March 8-11, 2017) ACM, NY, USA, 693-694.
- [25] Plucker, J., Beghetto, R. A., & Dow, G. 2004. Why isn't creativity more important to educational psychologists? Potential, pitfalls, and future directions in creativity research. *Educational Psychologist*, 39, 2 (June 2004), 83–96.
- [26] Porter, L. and Simon, B. 2013. Retaining Nearly One-third More Majors with a Trio of Instructional Best Practices in CS1, in *Proceedings of the 44th ACM Technical Symposium on Computer Science Education* (Denver, CO - March 6-9, 2013) ACM, NY, USA, 165-170.
- [27] Rich, L., Perry, H. and Guzdial, M. 2004. A CS1 Course Designed to Address Interests of Women, in *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education* (Norfolk, VA, USA - March 3-7, 2004) ACM, NY, USA, 190-194.
- [28] Riley, T. 1989. In C, Score and Performing Directions, *Celestial Harmonies*.
- [29] Simon, B. and Kinnunen, P., Porter, L. and Zazkis, D. Experience Report: CS1 for Majors with Media Computation, in *Proceedings of the 15th Annual Conference on Innovation and Technology in Computer Science Education* (Bilkent, Ankara, Turkey - June 26-30, 2010) ACM, NY, USA, 214-218.
- [30] Sloan, R. and Troy, P. 2008. CS 0.5: A Better Approach to Introductory Computer Science for Majors, in *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (Portland, OR, USA - March 12-15, 2008) ACM, NY, USA, 271-275.
- [31] Tew, A., Fowler, C. and Guzdial, M. 2005. Tracking an Innovation in Introductory CS Education from a Research University to a Two-year College, in *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education* (St. Louis, MO, USA - February 23-27, 2005) ACM, NY, USA, 416-420.
- [32] Tims, J., Zweben, S., Timanovsky, Y. and Prey, J. ACM-NDC Study 2016–2017: Fifth Annual Study of Non-Doctoral-Granting Departments in Computing, *ACM Inroads*, 8(3), pp. 48-62, ACM, NY, USA.
- [33] Tims, J. 2018. Achieving Gender Equity: ACM-W Can't Do It Alone. *Communications of the ACM*, February, 61(2), p. 5.
- [34] Trueman, D., Cook, P., Smallwood, S. and Wang, G. 2006. 'PLOrk: The Princeton laptop orchestra, year 1', in *Proceedings of the International Computer Music Conference (ICMC 2006)*, Michigan Publishing, 443–50.
- [35] Uludag, S., Karakus, M. and Turner, S. Implementing IT0/CS0 with Scratch, App Inventor for Android, and Lego Mindstorms, in *Proceedings of the 2011 Conference on Information Technology Education* (West Point, NY - October 20–22, 2011) 183-190.
- [36] Zweben, S. and Bizot, B. 2016 Taulbee Survey: Generation CS Continues to Produce Record Undergrad Enrollment; Graduate Degree Production Rises at both Master's and Doctoral Levels, *Computing Research Association*, <https://cra.org/crn/wp-content/uploads/sites/7/2017/05/2016-Taulbee-Survey.pdf>