How Software Works: Computational Thinking and Ethics before CS1

Andrew Scott
Department of Mathematics and Computer Science
Western Carolina University
Cullowhee, NC 28723
Email: andrewscott@email.wcu.edu

Scott Barlowe
Department of Mathematics and Computer Science
Western Carolina University
Cullowhee, NC 28723
Email: sabarlowe@email.wcu.edu

***Abstract*— It is widely acknowledged that many freshmen go to university without any prior grounding in computer science. Recent studies conducted in the US have shown that not only do high school students lack any exposure, but also they possess ill-conceived notions of what computer science is, a problem also affecting their parents, teachers and regional school superintendents. For incoming students, the extent to which computing technology impacts their daily lives is very likely at odds with the extent to which they have considered how software works and its ethical implications. The ability to critically analyze and consider related ethical consequences of computing is an important life skill for every twenty-first century adult and not just computer scientists. With this this in mind, this paper presents a unique CS0 introduction to programming and computer ethics for pre CS and non majors. It has been defined as a service course to promote digital literacy and to ensure that students appreciate what computer science is and its socio-ethical implications. The paper outlines the course and its content in detail, as well as providing quantitative and qualitative evidence of its benefit and appeal to female students.**

Keywords—Computer Science Education; ethics; programming; digital literacy,

I. Introduction

This paper presents a unique pre CS semester long course offering. The course aims to teach computational thinking and problem solving, and the ethical implications of software and its impact on our society, both nationally and globally. The programming component is delivered though hands-on problem solving tasks in the Scratch language [21]. The ethical component is delivered through student led presentations, debates, research and paper writing with a specific emphasis on current events. Since its introduction in the fall of 2013, the semester long course has been taught to ten classes over five semesters. In this paper we give details on the course design, its learning material, and teaching pedagogy and student outcomes.

Qualitative and quantitative data demonstrates the course's impact on its students and those graduating to the subsequent CS1 Java based objects first introduction to programming. Also of interest is the higher than average female enrollment compared with CS1. The findings are relevant to computer science instruction at university, community college and high school levels where a foundational introduction to computational thinking, program development and its ethical implications may be beneficial.

II. Background

A. Motivation

Computers are now a pervasive technology in our personal and professional lives. The ability to program is crucial to success in so many STEM subjects from chemistry, biology though to physics, engineering and medical fields. There is a wide body of research that shows that seldom do students arrive at university with any significant exposure to the technical aspects of computing and programming [26]. Programming is also perceived as difficult, as both an abstract and a precision intense discipline. The extent to which a beginner encounters errors is to them unusual and is known to debilitate their motivation and interest in the subject [5] [13]. Consequently, programming courses have far higher than average drop out and failure rates, which is a longstanding and global issue affecting the discipline [27] [2]. This is also a main driver behind the extensive research regarding novice programmer difficulty.

Globally many governments are now realizing the importance of teaching computational thinking in schools [4][23]. However, a proportionally large number of high schools still only teach computing as an office skill rather than a technical science with social consequences [26]. A recent study by Google surveying 1,673 k12 students, 1,685 parents, 1,013 teachers, 9,693 principals, and 1,865 superintendents across the United States found that many respondents did not possess a clear idea of what computer science was [26]. A predominant misconception was that CS includes creation of documents (a belief of between 63% and 75% of the demographic groups surveyed) and searching the internet (a belief of between 43% and 63% of the demographic groups surveyed). Likewise, our experience indicates that students have also not considered, or at the least have a very limited understanding of the ethical implications of the ubiquitous computing technologies that surround their lives.

Without prior experience and with tendencies toward deep misconceptions about the subject, how can new students make informed choices about studying a subject at university? If students find learning programming difficult, what will motivate them to continue? How will students know the consequences of the future products they will use or produce and the harm caused by badly designed hardware or poorly written and tested code? How will they place their technical knowledge in a broader social context as they progress?

For this reason our department identified a need for a pre CS1 course to introduce inexperienced new students to

computer science, programming and the wider ethical dimensions of computing. Our aim was to achieve this within a highly motivating and engaging context. In the next section we outline the reasons behind our choice to use the Scratch language at a university level. Following this we provide a background and rationale for the position that computer ethics is a general life skill.

B. Why CS0 and the Scratch Language

Even in the twenty-first century, in a typical introductory programming course students still build non visual command line centric programs. This is at odds with the graphically interactive nature of modern computing that the millennial generation has come to expect. A programming language can also distract greatly from the problem solving aspects of programming [11], where students end up focused more on the minutiae of syntactic representation than they do on problem solving and the development of a logically correct design. Therefore the choice of language needed the following attributes:

Small learning curve: *Learning the language should not distract from the problem solving aspects of programing.*

Provide transferable skills: *The language should provide features common to most programming languages such as sequence, selection, iteration, variables, functions and arrays.*

Visual and Interactive Programs: *The development of visually interactive programs should not require language features beyond that of an introductory programming course*.

For the reasons above, syntax intensive languages such as Java, JavaScript, C, C++ C# were undesirable. It is possible to make visually interactive programs in these languages, but this requires a significant amount of code or advanced language features beyond the scope of CS0. To reduce the overheads of the programming language, whilst facilitating the development of visually interactive programs (VIPs) it was felt that a dedicated pedagogic solution was required.

Over the past decade there have been prominent developments in the form of simplified programming environments that support the development of highly interactive visual programs. Examples are Alice [18], Greenfoot [13], Scratch [20] and its derivative Snap [16] language.

Alice employs a drag and drop programming interface and is the only one to support the development of VIPs in 3D. It has been shown to be effective when used within a CS0 context [3]. In Alice, programs are limited to the 3D models and landscapes provided, which limits the range of programs that can be built. Also, the mechanics of the 3D engine and object placement can take significant time and attention away from the development of the underlying program.

Greenfoot is a Java based development environment for novices and facilitates the development of simple visual games though the manipulation of Java Objects. This is achieved though the development of methods written as either Java code or an equivalent visual syntax called Stride. Unlike the other teaching solutions discussed in this section, Greenfoot embodies the object oriented rather than the imperative paradigm. Because of this the semantics of its program instructions are not strictly sequential. For example, a move(4) command followed by turn(3) will not move then turn the object; rather the two operations occur concurrently. Also, this cannot be achieved without the use of the core OO concepts of inheritance. Regarding our CS0 course, it was thought that this paradigm shift might distract from the development of problem solving skills in relation to sequence, selection and iteration.

Another consideration for the development of VIPs was media computation in the Python language as advocated by Mark Guzdial [7][6]. Media computation involves the development and manipulation of sound, pictures and video though code. This was initially trialled in the CS0 course described in this paper. While the on screen results were impressive, many students (predominantly the non-CS ones) did not appear to grasp the underlying concepts and often got lost in the syntax.

Scratch is an imperative visual programming language in which the need to type precision intense error prone code has been replaced by colorful, slot together, nestable and elastic blocks (see figure 1 three pages down). A program is constructed by populating a workspace (the stage) with 2D sprites and background imagery. For each sprite or background, users can develop one or more control scripts by dragging and dropping the desired language features into the desired sequence within the code space. Users can then type or select parameters to control the effect these blocks have on the sprite they are applied to. The benefit of this is to drastically reduce the learning curve of the language. Right from the start of each task, students can focus their attention purely on problem solving and algorithmic development, as the mechanics of the scene creation and language syntax represent minimal overhead.

Scratch supports the basic imperative programming constructs that appear in most programming languages including variables, assignments, decisions (if, if else), conditional operators, loops, arrays and custom functions. In addition, Scratch supports a comprehensive, yet constrained and consistent set of additional commands that can be applied to a sprite to provide motion, sensing, sound, concurrency, user control and remote procedure call capabilities. This allows Scratch to be applied to a limitless number of scenarios and problem types from games to animation to interactive learning. The range of programming constructs are sufficient to cover all of the concepts of a typical introduction to programming. Although Scratch is not an object oriented language, it does support the notion of objects in the way in which functions are called on the sprites (objects) and in how each sprite maintains its own state, size, position color, rotation etc.

Although Scratch was not intended for use at the college or upper high school level, an unintended consequence of its simplicity, flexibility and the clarity in which it conveys programming concepts has led to its use as a tool to convey the principles of programming at a university level, for example Harvard [29] and Norfolk State University [22]. Scratch 2, launched in May of 2013 also benefits from being entirely web based and accessible by any Adobe Air enabled browser. This represents a significant convenience over other such technologies which require downloading and installation.

C. *Why Computer Ethics in CS0*

Since 1987 the ACM / IEEE-CS joint curriculum task force has increasingly recognized and ensured the inclusion of computer ethics and related professional standards [9][25][10]. A 2008 study of 251 computer science programs in the USA determined that 88% do address computer ethics in their curriculum [24]. However, in our research of both the ACM and IEEE digital libraries we found no evidence of ethics being covered at the pre CS (CS0) level.

Since the millennium and in the wake of the internet revolution, technology has continued to permeate many aspects of the professional and social lives of modern technologically integrated societies. This trend is very likely to continue as personal computing technology evolves and ideas such as the Internet of Things [8] transitions from vision to reality. Today's students are what Mark Prensky has termed 'digital-natives'[18]. That is, they have grown up with digital technology and have likely spent their lives surrounded by web enabled hardware including computers, video games, smartphones and other media streaming devices. Online social interaction is also likely to be an integral everyday part of their modern lives.

For incoming students, the extent to which computing technology impacts their lives, is very likely at odds with the extent to which they have considered its ethical consequences. Most freshmen will be familiar with terms such as 'cyber bullying', 'hacking', 'online piracy' or 'cyber-crime', typically from news stories online, television or in the printed press; some may have even experienced these issues first hand. However, in our experience, few if any of our incoming students have ever considered such ethical issues within an educational setting. As a consumer, user, stakeholder or potential developer of such technologies, an appreciation of and ability to critically analyze and consider its related ethical consequences is an important life skill. We hypothesize that due to the ubiquity of computers, this holds true regardless of whether a student chooses to study computer science or any other subject. Therefore, such a course is beneficial and should be open to students of all disciplines.

We are a liberal arts focused college within a comprehensive master's degree granting university. At our institution incoming students with 0-15 hours of credit are required to take a liberal studies oriented freshmen seminar of three hours per week within their first year. This is also open to any student with 30 hours of study credit or less. The first year seminars are designed to support the ongoing success of incoming freshmen by acclimatizing them to the study skills and habits needed to succeed in higher education. The goals of the freshmen seminar are as follows:

- Learn about the importance of Liberal Studies in a university education;
- Consider how reasoning skills and communication skills are the foundations for life-long intellectual and professional growth;
- See that cultural, social, economic and political issues of a global society are not limited to one academic discipline or one profession;
- Discuss serious ideas and develop rigorous intellectual habits.

In addition, our freshmen seminars fall under our liberal studies criteria which defines the following goals:

- Locate, analyze, synthesize, and evaluate information;
- Interpret and use numerical, written, oral, and visual data;
- Read with comprehension;
- Write and speak clearly, coherently, and effectively;
- Critically analyze arguments;
- Recognize behaviors and define choices that affect lifelong well-being.

Using this list of requirements as a guide, a pre-CS course (open to all freshmen) was defined to encompass a stimulating and fun introduction to programming though Scatch, and ethics though in-class discussions, presentations, formal team debates, and position papers via the practice of research and collaboration.

Because of these agendum, the first year course also serves more general goals, helping new students gain the study habits crucial to success in higher education i.e. writing, research, critical analysis, collaboration, presentation, and problem solving. This agenda brings a diversity to the instructional techniques used to deliver the course. In the next section we elaborate on the structure and delivery of this course offering.

III. THE COURSE

The course was defined as a service course to promote digital literacy, to ensure that students appreciate what computer science is and its socio-ethical implications. The course was not specifically created as a recruitment course. However, it is acknowledged as a potential positive side effect.

Since the fall of 2013 the CS0 course has been delivered by four faculty members to 212 incoming students with between 0 to 30 credit hours. It is one of several liberal studies courses freshmen can choose. Class sizes are typically between 20 to 27 students in number with on average two sections taught per semester (due to interest and high demand). To provide sufficient time for tasks, each class meeting lasts 75 minutes and meets twice weekly. This totals at 30 instructional periods over one semester.

A core goal of our instructional technique was to maximize student engagement in all activities. Therefore, lecturing activities were minimized to at most 10 to 20 minutes, with most class meetings featuring no lecture component.

The course is split equally into two main components. First is a hands-on introduction to programming and problem solving though pair programming in Scratch. Second is a highly collaborative computer ethics component featuring student presentations, formal team debates, research skills, and the production of team and solo ethics papers. The course culminates by combining the two main themes into one final project using their new found programming skills to present a current ethical issue (of their own choice) though a game, or interactive learning application of their own design in Scratch.

Below we outline the programming, ethical and final projects and exams of the course in more detail. Although there are slight variations among instructors, this paper reports a successful delivery structure most commonly found in course offerings.

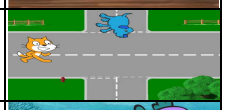
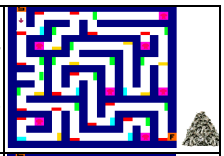
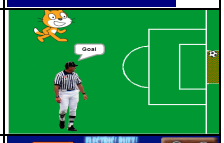
A. Programming Component

In delivering the programming component, we were very mindful of the heritage of the Scratch language, which was developed at MIT's media lab. It is heavily influenced by the constructionist learning theories of Seymour Papert [16] who at MIT brought the logo language to many classrooms in the 1980's. Constructionist learning is about learning though discovery. Students learn though participation in project based learning rather than lectures. In doing so they apply knowledge they already have to the development of new skills.

Throughout the course, students are introduced to the concepts of imperative programming concepts, though 15 fun and engaging problem solving tasks of incremental difficulty. Pair programming [28] is used to stimulate collaboration and the communication of ideas between peers. The tasks start with basic animation, then quickly move on to game based challenges. For example, conditional logic is taught via a maze navigation challenge whereby color blocks in corners and junctions are sensed by a sprite and used to steer it through the maze to avoid the poison and win the prize at the exit.

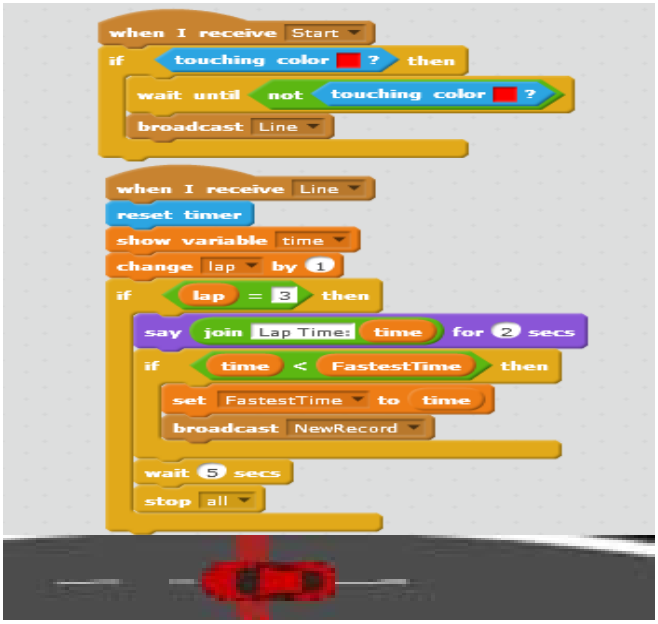
At all times instruction promotes the use of proper programming terminology and the development of conceptual understanding though problem solving. Each concept (for example, conditional logic) is introduced by the lecturer on the white board and via the in-class projector to show the concept in action and in isolation in Scratch. Typically this is done by applying the concept to a small number of sprites. During this time, the students are expected to repeat these actions on their own workstations. The instructor uses this as an opportunity not just to show concepts but to ask leading questions to elucidate the consequences of certain actions from the students, for example by deliberately misapplying conditional logic. Following the introduction of a concept, the students are then given a worksheet, provided as a print out and accessible from the course website, and are tasked with applying the concept introduced in a problem solving task. Below is an overview of the 15 programming problems the students will solve over the semester, and the skills and concepts covered within them:

TABLE I. IN CLASS PROGRAMMING PROJECTS

Task A: Cat follows wall: *Skills: program statement sequence, sprite movement and sound.*	
Task B: Cat follows Spiral *Skills: program statement sequence, movement and sound*	
Task C: Cat Meets Dog *Skills: basic conditional logic though sensing.*	
Task E: Fish Tank *Skills: repetition though looping, defining 4 unique aquatic behaviors*	
Task F Cat Meets Dog Again *Skills: combining loops and conditions*	
Task G: Maze *Skills: User interaction though keyboard control to escape the maze.*	
Task H: Maze Logic *Skills: problem solving using four color sensing and conditional logic to define an algorithm to avoid traps and escape the maze.*	
Task I Maze Logic 2 *Skills: The use of logical operators to develop a six color sensing maze escape algorithm. Task involves two levels, a key and backtracking.*	
Task J: Soccer *Skills: Uses conditional logic to form bounding box and remote procedure calls for sprite interaction.*	
Task K: Buzz: *Skills: Mouse interaction and multiple remote procedure calls to build a classic game to test dexterity, steadiness and patience.*	
Task L: Frogger *Skills: User defined variables, larger more complex scripts and multiple sprite interaction are used to implement this classic video game.*	
Task M: Frogger 2.0 *Skills: Employs screen switching, advanced sprite manipulation, with additional conditions and variables to take Frogger to another level.*	
Task N: Drive *Skills: Utilize the system clock and model acceleration to produce this single lap driving simulation. Don't hit the barrier!*	
Task M: Drive 2.0 *Skills: Uses variables and conditions to deduce and retain the fastest time and lap of a three lap race. Uses functional decomposition. Pull skids.*	
Task O: Destruction Derby *Skills: A two player demolition derby employs smart collision detection and functional decomposition. It is the largest and last of the projects.*	

The graphical nature of these tasks may lead the reader to presume that the tasks are over simplified. To the contrary, many of the tasks employ an algorithmic complexity beyond that which could be achieved by the equivalent number of instructional hours in our follow on introduction to programming in Java (CS1). For example, consider tasks N and M. The race timer must start when the front of the car intersects with the start line and stop when it intersects with it once again on the next lap. However, care must be taken because as the car crosses the line, intersection will be detected at every frame of animation, potentially stopping the timer straight after it has started. It is problems such as these that require the students to use the programming concepts with a detailed level of attention to their run time semantics. Figure 1 is an example of the algorithm for deducing lap times in task M which represents less than a third of the required code to solve this task.

Fig. 1. Scratch Algorithm for race timing.



Task I is similarly involving, whereby users must develop a six color detection algorithm responsive to color changes on the maze floor (once a key is collected) to programmatically steer the character safely to the exit. The solution requires multiple sets of conditions and the application of logical operators. This task serves very well as a concrete introduction to Boolean logic, as the other tasks serve to introduce each concept within a highly engaging problem solving context. In the following chapter the ethical component is discussed.

B. *Computer Ethics Component*

The ethical component starts with an introduction to ethics given as an involving discussion outlining what ethics is and how computer ethics is a branch of applied ethics. Also covered is computer ethics in a historical context, discussing the emergent need for computer ethics as ethical problems arose as a consequence of computing technology. For example:

- 1960s Computer crime and misuse,
- 1970s First computer viruses
- 1980s Pirated software and violence in video games
- 1990s Public use of the internet
- 2000s File sharing, social media and internet privacy
- 2010s 3D printing and government surveillance

We also discuss professional standards of integrity such as the ACM's code of ethics [1] and Computer Ethics Institute's Ten Commandments [19]. Other important concepts discussed are ethics and the law in which examples of situations legal but not ethical and ethical but nor legal are discussed.

In addition, we discuss the application of the greater good, when faced with opposing moral standpoints. These elements are important, as often students tend to think in black and white, when in reality many ethical situations are not clear cut. A comment received from students at least once per semester is that they are not used to there being multiple opposing but equally valid ethical considerations when faced with dilemmas.

A particular feature of most classes is to start the lesson off by discussing current ethical issues that are being raised in globally oriented news websites such as news.bbc.co.uk and cnn.com. Finding recent computer ethics news did not present a challenge for any task; there was a plentiful supply of topics on popular news sites and within academically oriented sources such as ACM portal and IEEE explore.

1) *One Slide Ethical Dilemma*

The first prominent ethical task is the presentation of an ethical dilemma, which is used as a way to get students to view an ethical issue from multiple perspectives. Following several examples, the students were asked to produce and present in pairs or threes, a one slide, three minute ethical dilemma with an additional two minutes for class questions and discussion. These are presented around the end of the second week. This is also a convenient way to ease the students into speaking. The topics were something the students had to choose (subject to lecturer agreement) and research. Examples are: human enhancements, social media anonymity, government surveillance, software patents, hacktivists, and technology versus jobs. The students were graded using a simple rubric covering, slide content, slide style, the use of supporting imagery, the validity of the dilemma, and the clarity with the dilemma is conveyed. At this early stage in the course, the students were not graded on how they handled questions.

2) *Ethical Debates and Team Paper*

At the beginning of week three, the students were split into teams of three (and one team of two when required) and tasked with choosing topics for their ethical debates. This began with an in class discussion covering some recent ethical events in computing, of which there are always many. For example, one memorable topic was Shodan [14] an online search engine that lets one browse and connect to any device connected to the internet which has not had its default password changed. Each ethical topic was chosen and worded so that arguments could be made for and against the topic. Therefore for Shodan HQ, the following statement was made: Shodan HQ makes the internet more, not less, secure.

Once there were more valid topics than teams, the students were asked to rate their preferences. This was used to assign one ethical topic to two teams, one for and one against. The teams did not choose what side of the argument they wanted; it was assigned purely at random by tossing a coin. Having the teams argue a point they do not necessarily agree with, takes them out of their comfort zone and forces them to see an ethical issue from another perspective. It is also a better test of their research debating skills. Each debate lasted roughly 35 minutes, including time for questions, which permits two teams per class period. With 26 students that is nine debates over four to five class periods. Table II outlines the procedure used in the debate.

In addition to participating in the debate, each team is expected to produce and turn in a draft debate paper of at least 1200 words on the day of their debate. The paper must contain: the arguments the team will use in the debate, the counter arguments they anticipate the other team will make, and the responses. To aid them, the students are given anonymized examples of previously written debates from graduated students (of a prior more senior ethics course). In

addition, teams are encouraged to go to the university's writing and learning support center to seek assistance when required. To mitigate the problem of teams scheduled later having more time to produce their work, the students are allowed to turn in a revised copy of their draft, following feedback from the lecturer by an agreed date. Should teams submit a revised version the grade is split 50/50 between the draft and final versions. The students are expected to identify all parties, the effects on those parties and the relative weights of different effects, and use this to come to a well reasoned conclusion. The papers must also contain at least three citations. An alternative to debates also used in CS0 is team presentations (~15 minutes). These have to include the historical origins of the topic, related legislation, all pros and cons, a case study, and then end with the group's consensus if it was generally a pro or a con and questions from the class.

TABLE II. THE FORMAL DEBATE PROCESS

Debate Part	Team	Minutes	Purpose
Opening Statement	Affirmative	1.5 – 2	Present a logical and convincing argument in support of your team's position.
	Negative	1.5 - 2	
Conference		<= 3	The members of each team confer on their responses.
Rebuttal and conference repeated three times			
Rebuttal	Affirmative	1.5 – 2	Present an argument refuting the argument that the other team has presented.
	Negative	1.5 – 2	
Conference		<= 3	The members of each team confer on their responses.
Closing Statement	Affirmative	<= 2	Summarize in a convincing and clear manner your team's argument and how you refute the argument of the other team.
	Negative S	<= 2	
Answer Questions	Both Sides	Variable Amount of Time	Present an answer to the question or a rebuttal of the other team to a question, make a convincing argument, and present it clearly.

3) Solo Paper

In addition to a team ethics paper, the students are tasked with producing a solo paper, either after the mid-term or at the start of term for hand in at the mid-term. Once again, the topic is of the student's own choice. This time, the student is given more leeway with what standpoint they wish to take, either affirmative, negative or balanced.

4) Imparting Research and Referencing Skills

After the first semester it was realized that the students needed more support and encouragement to locate and cite proper sources of research. It was clear that many did not know what constituted a robust citation with blogs, forums and personal opinion featuring in many papers. They also needed more guidance in preparing attractive, well laid out slides. Consequently, an additional class period was devoted to research and referencing as well as what constitutes good and bad presentation slides.

C. Final Project

The course culminates in the production and presentation of a final project which is assigned in pairs at the midterm. The final project brings together both the programming and ethical components. The students are tasked with building a Scratch program to highlight a particular ethical issue. This could be a game or an interactive learning application. In the last week of class, each pair is expected to present their program in class. This project is graded based on the complexity and quality of the underlying code solution, how well the students communicate the technical aspects of the project, the level of interactivity and how well it addresses the ethical issue assigned to the team. For both students and faculty, this aspect of the course is very rewarding.

D. Exams and Tests

Depending on the time available, the course included one or two tests and a final exam. The tests and exams follow a two part format. The first part is closed notes and closed book. The students have to answer 10 to 15 questions which incorporate a mixture of ethics and Scratch based programming questions. The ethics questions generally test the student's understanding of core ethical concepts by having them explain the concept. Also included are short ethical scenarios in which the students have to examine the stakeholders and ethical consequences. The programming questions follow the form of a typical programming exam, but use Scratch syntax instead of code. For example, the student is shown some code and the inter-related sprites. They then might be asked to explain the outcome of the code, or to suggest modifications to change its behavior. In almost all of the programming questions we try to elucidate core language independent imperative programming concepts rather than Scratch specific terms to make the experience transferable.

The second part of the test is a programming challenge, whereby we describe the desired outcome and provide the relevant media (images and sound files) via the course website. Students then submit their solutions to our virtual learning environment (Blackboard). Tests are focused on recent topics performed during term time and run for approximately 70 minutes. The exam follows a similar format but is two hours long and tests knowledge from the entire course. Table III outlines the proto-typical in class schedule over 15 weeks.

TABLE III. THE STRUCTURE OF IN CLASS ACTIVITIES

wk1	Course Syllabus and policies, Sequential logic (tasks, A and B). Intro to ethical concepts and computer ethics origins
wk2	Current ethical issues and dilemmas discussion, Conditional logic (task C), Choose pairs' topics for 1 slide dilemma (task D)
wk3	One slide dilemmas presented, Loops (task E), CS ethics in the news (discussion)
wk4	Debate teams make list of preferred topics, Debate topics and dates assigned to teams, Conditional Loops and communication (task F)
wk5	Writing and researching debate and paper, Research, referencing and good presentations, Sensing and user interaction via keyboard (task G)
wk6	Team debates (two teams), Nested conditionals and sensing (task H)
wk7	Team debates (Two teams), Complex maze navigation algorithm (task I)
wk8	Team Debates (One teams), Remote procedure calls (task J), Test One
wk9	Team debates (Two), Mouse interaction (ask K)
wk10	Variables, memory and data storage (task L), Pairs and topics for final projects, Solo paper topics chosen, Team debate papers due
wk11	Handling complexity (task M), Ethical discussion historical issues
wk12	Timers, variables, and acceleration (task N), Global ethical news discussion
wk13	Advanced logic & functional decomposition (Task O), work on project, Ethical discussion on software
wk14	Two player large-scale programing (task O), Working on project, Test 2
wk15	Final project presentations, Topic review

TABLE IV. ENROLLMENT TOTALS FALL 2013 TO FALL 2015

	CS0 (this course)	CS1 (programming1)	Institutional enrollment
Females	75 (30%)	27 (13%)	56%
Males	179 (70%)	181 (87%)	44%
Total	254	208	100%
Failed or withdrew	20 (7%)	35 (17%)	10% (freshmen only)

TABLE V. CLASS SECTIONS AND ENROLEMENT OF THE CS0 COURSE

	Fall 13	Spring 14	Fall 14	Spring 15	Fall 15
Section1	30	22	25	23	27
Section2	30	18	24	24	
Section3	31				

TABLE VI. EVENTIAL STUDENT MAJOR AFTER CS0

	Males	Females	All
CS	37	5	42 16.54%
CIS	20	0	20 7.87%
Non CS	112	70	192 75.59%

TABLE VII. COMPARING CS STUDENS WITH AND WITHOUT CS0

Group	Took CS0 (20)	Not Took CS0 (33)
Pass (>= C)	62% (197)	70% (23)
Fail <= D and Withdraw	38% (123)	30% (10)

TABLE VIII. QUANTATIVE RESULTS

No	Statement	Rsp	%	DS	D	U	A	SA
1*	I would like to work in computer science.	23	74	4	3	2	6	10
2	CS0 positively influenced my interest in computing (as a whole) as a vocation.	23	73	2	2	1	9	9
3*	Programming is an interesting subject	18	85	0	0	1	9	8
4*	CS0 positively influenced my interest in programing as a vocation.	17	84	0	0	1	9	7
5*	In CS0 I gained an understanding of programming concepts such as variables, loops and conditions and how they are used.	18	76	1	1	2	6	8
6*	CS0 enhanced my problem solving skills.	18	74	0	1	6	4	7
7	Before CS0 I had not studied computer ethics.	23	88	0	0	0	11	11
8	CS0 enhanced my perceptions and appreciation of computer ethics and its implications and importance.	23	68	1	0	8	9	5
9	CS0 Enhanced my interest in computer ethics	23	61	1	3	7	9	3
10	Computer ethics is something every computing student should be taught and made to consider.	23	78	0	0	7	6	10
11	I found the ethical debates / presentations were a worthwhile experience.	22	61	0	6	5	6	5
12*	I found the programming tasks and projects were a worthwhile experience.	17	81	0	1	2	6	8
13*	For me programming was the most important and worthwhile aspect of CS0.	18	82	0	0	5	3	10
14	For me ethics was the most important aspect of CS0.	22	47	3	6	7	3	3
15	CS0 contributed positively to my studies at WCU.	22	75	1	3	0	9	9
16	CS0 enhanced my ability and comprehension in subsequent / concurrent programming courses.	18	64	0	3	6	5	4
17	CS0 enhanced my abilities in subsequent ethics courses.	23	57	2	1	11	7	2

IV. EVALUATION

The results presented in this section represent our initial analysis of the data collected. As this paper is authored, additional survey data continues to come in.

A. Enrollment and Impact

One statistic that demonstrates the perceived worth of this course is in enrollment (table IV and V). Since the course's inception in the fall of 2013, a total of 254 incoming students (179 males and 75 females) have taken the course. Table VI shows data retrieved from our enrollment records and demonstrates that of these 42 went on to study computer science, 20 studied computer information systems, and 70 ended up in non computing subjects. Studies have shown that female enrollments are typically around 12% [21]. As table IV shows, the female enrollment in CS0 is much higher than is currently typical for a computer science course. Another interesting statistic is the dropout and failure rate of the course; it is lower than both our CS1 introduction to programming and our institutional average for Freshmen.

The CS0 course was not intended as a recruitmant tool for computer science. However, if it better prepares students for CS1 this will affect the way we advise incoming students. Following the introduction of the new course, the grades of CS1 students were divided into two groups, those who took CS0 and and those who did not. The results in table VII show that students who took CS0 are 8% more likely to pass the course. Taken at face value, this difference may seem insignificant. However, one must asknowlege that CS0 students are, on the whole, less experienced than those starting at CS1. Students starting in CS1 are far more likely to have either transfer credit from another post secondary institution, technical computing experience or enter as high acheivers in STEM subjects. CS0 students simply have less experience or demonstated ability. Without CS0, this cohort is at risk of performing worse, not better than their peers. Therefore the 8% difference is significant and indicative that CS0 is helping address retention and comprehention issues in CS1.

B. Student Reactions to the Course

The students who had completed the course anywhere between fall 2013 and fall 2015 (inclusive) were asked to complete an emailed questionnaire. It consists of closed rating questions (Likrert), and open free response questions so students could reflect on the experience and relate it to their current studies. The respondents were requested print out the questionnaire, complete it and submit it by hand. A free drink and small snack were given as incentives, but students were warned to be honest and truthful when responding. The survey was released in early April 2016 and currently has 23 student respondents, 18 CS, 4 non CS and one who is undeclared. There are 17 males and 6 females. This is less than we had hoped, but this still provides valuable insights, it represents the number of students that typically make up one CS0 class.

The responses to the rating questions are shown below in table VIII. These were 5 option Likert scales ranging from strongly disagree to strongly agree (where U is undecided). The responses were then converted to a percentage of the maximum possible score (all strong agreements), where strong disagreements scores 0, disagreements scores 1, undecided scores 2, agreement scores 3 and strong agreement scores 4. Therefore, a percentage score of 100 indicates total agreement, 75 indicates agreement, 50 indecision, 25 disagreement and 0 total disagreement. Questions marked with a * have fewer respondents because five of those replying took an initial section of the course in which Python was trialled and not Scratch. Other small variations were where students had skipped a question. To gain qualitative responses the students were asked four open response questions as follows.

- With respect to programming, what aspects of CS191 made the biggest impact on you (if any)? State why.
- With respect to computer ethics, what aspects of CS191 made the biggest impact on you (if any)? State why.
- What overall aspect of CS191 made the biggest impact on you and why?
- Your most favorite and least favorite aspects of CS191 and why?

The quantitative results show that the Scratch programming component was viewed very favorably by the students. This correlates with the open questions in which the students were asked to describe the biggest impact or their favorite aspect within the course. Of the 18 respondents who took the course when Scratch was offered, 18 named it or the related activities as their favorite aspect. For example:

"Had me decide to major in CS" (male)

"The biggest impact was programming because I discovered I could do almost anything though it" (male)

"Scratch taught me about coding without totally drowning me in new information" (female)

"Being able to make an actual game by the end of the course is one of my favorite achievements" (female)

Five of the 18 who used Scratch specifically indicated problem solving as its biggest impact, with five using the term directly and the others mentioning loops, statements, logic and blocks (Scratch's implementation of functions).

However, not all students appreciated Scratch; one student cited Scratch as the worst feature, with three others requesting a more conventional language. Initially in class, some students verbally expressed concern that Scratch was not a proper language. However, these fears soon abated once many students realized how versatile and fun it could be.

Other respondents seemed to note the social aspects of the course and in particular pair programming. for example:

"The atmosphere when an entire class works on a project" (male)

"It was great working with other CS majors and getting to know them"(male)

"Being social and making friends though common interests and programming. I feel as though pair programming makes programming more interesting" (male)

These points correlate with our observational findings in which the classes appeared studious yet light hearted and very sociable. This can also be attributed to the age of the students, all of whom were young adults.

Regarding the ethics component, the responses are generally positive. In particular, question 7 aligns with our initial assumptions that students have not previously considered computer ethics within a learning context. Question 10 shows that students believe the topic of ethics is an important part of the CS curriculum. When asked to state the most impactful aspects of the ethics component, 13 respondents had specifically positive comments. Five respondents cited particular ethical topics of interest that were discussed in class for example, security, privacy laws, Edward Snowden, data mining, and drones. Seven others commented on how the course raised their awareness of computer ethics, for example:

"I had not really thought about there being ethical issues so it kinda opened my eye that there were some" (female)

"It taught me to be conscious about what I do or say online" (female)

"Learning about computers and how they can be used for good and evil was very impactful. For later computer science ethics classes. Learning about data mining was interesting." (male)

Two respondents stated debates as a particular interest. One mentioned the final project as the most impactful ethics component, and another mentioned the team dynamics during debates. Our own observations corroborate these points and showed that on the whole, the debates and program presentations were, for both the presenter and audience, motivating and engaging experiences. However, on the occasion when a team is not prepared it is generally very evident. One student commented that his least favourite aspect was 'listening to unprepared debates'

However, the ethics results were not all positive. Four students indicated that the ethical component had little or no impact, and one failed to remember anything specific they liked. Two students cited written exam questions as their least favourite element. It is also worth noting that no one cited the solo or team papers as impactful, or enjoyable. Three respondents indicated significant levels of discomfort in writing the papers. This aligns with our experiences grading their work. It was clear that writing and researching a >= 1200 word paper was a task that some students found challenging, whilst some others produced rushed and ill thought out work. However, on the whole, students did well with written tasks. The cumulative results of the authors' own paper grading show that roughly 64.5% were graded at A or B, 25.5% at C or D, and 10% failed to submit or get a passing grade.

Question 14 achieved a mild disagreement. Currently, the authors are unsure why this response is so low given the perceived buy in from students we observed in ethical discussions, presentations and debates. Without a more fine grained analysis, it is presumed that students may be contrasting this question against the previous one because of the students' high opinions of the Scratch programming. However, it may also correlate with the discomfort felt in paper writing.

V. Conclusions

This paper outlined the need for a highly motivating and visually interactive introduction to programming at a pre-cs level. It also stressed the importance of teaching the socio-ethical impact of computing to all students. Given its widespread ubiquity and continual permeation within the lives of all students, as computing professionals, teaching this is our social responsibility.

This paper's contribution was to define a successful CS0 course that combines both a foundational introduction to programming and computer ethics. It was also to provide a rationale for doing so as well as evidence of its benefit. An additional aim was to assist others in the development and delivery of a similar course.

The ideas presented in this report are supported by consistently high enrollments of both male and female students, by the fact it has been taught 10 times since its inception and by the quantitative and qualitative analysis of the students' retrospective opinions and by our personal account.

Going forward, we aim to continue collecting data and intend to conduct a more fine grained analysis with additional questions and focus groups, and by correlating responses with the students' course grades and GPA. One plan is to ask specific pre and post course questions to examine how the course changes the students' perceptions of computer science and its ethical dimensions. Another potential area of interest is the female enrollment. What is it that is encouraging more females to enroll and how can we use this to leverage sustained interest? The suitability of some or all of this material at the high school level is also a future consideration. It is very clear that Scratch is not only very motivating, it places the core programming concepts within the heart of problem solving.

The Scratch programming task worksheets and media resources discussed in this paper are available at the following location: http://agora.cs.wcu.edu/~ascott/cs0

REFERENCES

[1] ACM Council, ACM Code of Ethics and PRoesional Conduct, 1992, ACM, New York, NY, Accessed April 2016: https://www.acm.org/about-acm/acm-code-of-ethics-and-professional-conduct

[2] J. Bennedsen,. & M, Caspersen, Failure Rates in Introductory Programming, ACM SIGCSE Bulletin, Volume 39 issue 1, 2007, ACM, Ney York - NY - USA, pp32-44.

[3] J. Cordova, V. Eaton and K. Taylor Experiences in Computer Science Wonderland, A Success Story with Alice, Journal of Computing Sciences in Colleges: Volume 26 Issue 5, 2011, Consortium for Computing Sciences in Colleges , pp 16-22.

[4] Computing in the national curriculum (UK), Computing At School, British Computing Society Academy of Computing, London, UK, 2014, Accessed April 2016: http://www.computingatschool.org.uk/data/uploads/CASPrimaryComputing.pdf

[5] E. Dunnican,, Making the analogy: Alternative delivery techniques for first year programming, Proceedings from the 14th Workshop on the Psychology of Programming, Interest Group ,PPIG, 2002, Uxbridge - UK, pp 89-99.

[6] M. Guzdial, Introduction to Computing and Programming in Python (4th ed), 2016, Pearson, New York, NY, ISBN-13: 9780134025544

[7] M. Guzdial, A Media Computation Course for Non Majors, ACM SIGCSE Buletin, Volume 35, Issue 3, 2003, ACM, New York, NY, p100-108.

[8] J. Höller, V. Tsiatsis, C. Mulligan, S. Karnouskos, S. Avesand, D. Boyle: From Machine-to-Machine to the Internet of Things: Introduction to a New Age of Intelligence. Elsevier, 2014, ISBN 978-0-12-407684-6.

[9] C. Huff, C. Martin, Computing Consequences: A framework for teaching ethical computing, Communications of the ACM, Volume 38, Issue 1, 1995, ACM, New York, NY, pp75-84.

[10] IEEE-CS/ACM Joint Task Force on Computing Curricula. , Computing Curricula 2001 Computer Science Final Report IEEE-CS/ACM Joint Task Force on Computing, 2002, Los Alamitos, CA: IEEE Computer Society.

[11] C. Kelleher and R. Pausch, Lowering the Barriers of Programming: A taxonomy of programming environments and lanugges for notive programmers. ACM Computing Surveys (CSUR), Volume 27, Issue 2, ACM, New York, NY, pp 83-144.

[12] M. Kölling, Greenfoot Interactive Java Development Evironment V3, 2013 Computing Education Group, University of Kent, Cantebury, UK, Accessed April 2016: http://www.greenfoot.org

[13] M. Lee , A. Ko, Personifying programming tool feedback improves novice programmers' learning, Proceedings of the seventh international workshop on Computing education research, 2011, Providence, Rhode Island, USA.

[14] J. Matherly, Shodan Computer Search Engine, 2009, Accessed April 2016: http://shodan.io

[15] J. Mönig, Snap Language (formerly Build Your Own Blocks), University of California Berkeley, Berkeley, CA, Accessed April 2016: http://snap.berkeley.edu

[16] S. Papert and I. Harel, Situating Constructionism, In Constructionism I. Harel ed, 2001 Ablex Publishing, ISBN-10: 0893917869, Accessed April 2016: http://www.papert.org/articles/SituatingConstructionism.html

[17] R. Pauch and C Kelleher, Alice Programming Environment and Language, Carnegie Mellion University and University of Virginia, Acessed April 2016: http://alice.org

[18] Prensky, Marc "Digital Natives, Digital Immigrants". On the Horizon Volume 9, Issue 5 2001, MCB University Press, Bingly, UK, pp 1–6. Retrieved April2016:http://www.marcprensky.com/writing/Prensky - Digital Natives, Digital Immigrants - Part1.pdf

[19] C. Rammon, Ten Commandments of Computer Ethics, Computer Ethics Institute, Washington DC, USA, Accessed April 2016: http://computerethicsinstitute.org/publications/tencommandments.html

[20] M. Resnick, Scratch Programmig Language version 2, 2013, MIT Media Labm Massechusets Institute of Technology, Cambridge, MA, Accessed April 2016 http://scratch.mit.edu

[21] B. Richards, Representation of woman in CS: how do we measure a program's success, ACM SIGCSE Bulletin, Volume 41, Issue 1, 2009, ACM, New York, NY, pp 96-100.

[22] M Rizvi, T Humphries, A Scratch-based CS0 Course for At-risk Computer Science Majors, In Proc Frontiers in Education Conference, 2012, , IEEE , Seattle, WA.

[23]M. Smith, Computer Science for All, The White House, 2016, Washington – DC, Accessed April 2016: https://www.whitehouse.gov/blog/2016/01/30/computer-science-all

[24] C. Spalding, K. Soh and C. Asorge, Ethics training and decision-making: do computer science programs need help?, ACM SIGCSE Bulletin, Issue 35, Volume 1, 2008, ACM, New York, NY, pp153-157.

[25] A. Tucker, Computing Curricula 1991, Communications of the ACM, Volume 31, Issue 6, 1991, ACM, New York, NY, pp 68-84.

[26] J. Wang, H. Hong, J Ravitz and S Moghadam, Landscape ok K-12 Computer Science Education in the US, Perceptions, Access and Barriers. In Proc 47th SIGCSE ACM Technical Symposium in Computer Science Education, 2016, Memphis, TN, ACM, New York, NY.

[27] C. Watson, F. Li, Failure Rates in Introductory Programming Revisited, In Proceedings of the 2014 Conference on Innovation and Technology in Computer Science Education, Uppsala Sweden, ACM, 2014, New York, NY, pp 39-44.

[28] L. Williams, Eleven Guidelines for implementing pair programming in the classroom, In Proc AGILE 2008 Conference, 2008, Toronto Canada, IEEE Computer Society, pp 445-452.

[29]] U. Woltz, H. Leitner, D. Malan and John Maloney, Starting with Scratch at CS1, ACM SIGCSE Bulletin, Volume 41 Issue 1, 2009, ACM, New York, NY, pp2-3.