”Keep your eyes on your own paper” - academic dishonesty in the era of online homework assistance

Dr. Kenneth Reid, Virginia Tech

Kenneth Reid is an Associate Professor in Engineering Education at Virginia Tech. He is active in engineering within K-12, serving on the TSA Board of Directors. He and his coauthors were awarded the William Elgin Wickenden award for 2014, recognizing the best paper in the Journal of Engineering Education. He was awarded an IEEE-USA Professional Achievement Award in 2013 for designing the nation's first BS degree in Engineering Education. He was named NETI Faculty Fellow for 2013-2014, and the Herbert F. Alter Chair of Engineering (Ohio Northern University) in 2010. His research interests include success in first-year engineering, engineering in K-12, introducing entrepreneurship into engineering, and international service and engineering. He has written texts in design, general engineering and digital electronics, including the text used by Project Lead the Way.

Max Mikel-Stites, Virginia Tech

Max Mikel-Stites is pursuing master's degrees in engineering mechanics and mathematics at Virginia Tech. He studies the biomechanics of hearing in parasitoid flies and is passionate about the physics of Marvel superheroes and scientific communication. His general research interests include biological modeling on both organismal and population scales, biological physics, and agent-based modeling. He graduated with degrees in applied mathematics and physics & astronomy from the University of Rochester.

“Keep your eyes on your own paper" - academic dishonesty in the era of online homework assistance

Introduction:

Nearly every engineering program has an introductory programming course or a course in which they introduce computer programming. A large mid-Atlantic university includes MATLAB programming in their Introduction to Engineering course sequence as is typical [1]. In these courses, programs are often simplistic and very straightforward; beginning programs may be 10-20 lines of code, and problems may not have multiple solutions. As the course progresses, the programming assignments become more complex and students begin to have an opportunity to solve problems using different approaches. In this particular course sequence, the programs start to increase in complexity after an introduction to functions, and there are a number of fairly complex programming assignments in the second semester course.

Students are introduced and held to the policies as defined in the Undergraduate Honor Code, which clearly define cheating:

> Cheating includes the intentional use of unauthorized materials, information, notes, study aids or other devices or materials in any academic exercise, or attempts thereof. … Acquiring answers from any unauthorized source in completing any assigned work. For assigned work, unauthorized sources include, but are not limited to, working with another student on a project that is to be completed individually, copying solutions from an online source or solutions manual, using the services of commercial term paper companies, or purchasing answer sets to homework assignments. [2]

Course coordinators of this course selected assignments to submit to MoSS, an online service which measures the similarity of the code contained in any two submitted files. The MoSS algorithm is such that if a similar block of code appears enough times in submitted files, it is assumed that that code must have been supplied to the students; in other words, if most of the students typed a similar block of code, it must have been given as part of the assignment [3].

Coordinators noticed idiosyncrasies in one assignment in particular, the “Trajectory” assignment. The assignment task was to calculate the trajectory of a launched object using simple kinematic equations. In particular, multiple students had identical comments, containing identical phrases such as “calculate the lunch angle,” output the “horizentol velocity,” and used a seldom-seen, and in this case unnecessary, MATLAB command, uint64. Investigating further, the authors found almost 100 cases of code that showed signs of academic dishonesty in this assignment alone, with clusters of students with nearly identical structure or comments, and often identical errors. Most of the unusual solutions were found on online homework assistance internet sites.

This paper will document a case study of the process of identifying cases of widespread, detected academic dishonesty in multiple MATLAB assignments and describe the aftermath: identifying and notifying approximately 10% (188 students of 1861 enrolled) of the students in a large course. Finally, given that the goal is to prevent academic violations and stress the importance of students submitting their own work, the authors will address the failures of traditional methods of detection and prevention of academic dishonesty.

Background:

For many of us, the warnings about academic dishonesty in our undergraduate programs took the form of exhortations to "keep your eyes on your own paper" and restrictions on baseball caps, baggy jackets, and water bottles, perhaps with elaborately produced fake labels. Proctors were alert for wandering eyes, craning necks, and cupped hands held awkwardly by a student's side. Copying on tests meant an overly casual sideways glance and wandering attention. Copying on homework materials often depended on peer social networks, and the perception of academic dishonesty within those networks.

Academic misconduct is widely known on college campuses. Most institutions have an office of academic integrity and policies in place to attempt to curtail multiple types of cheating. Studies have shown that implementing a formal honor policy can deter academic dishonesty [4,5] and effective communication of honor code policies tend to reduce the frequency of cheating [6-9].

More recently, the available avenues for academic dishonesty have exploded. In addition to more traditional approaches, smartwatches can be programmed with entire crib sheets, cell phones can silently take high resolution pictures and display the results, class materials are posted wholesale online, and "homework assistance" websites answer student questions in real time. One significant emerging difference between the new and old resources is a reduced involvement of students' personal social networks. Online resources are quick, simple, and anonymous. As a result, rather than accepting the risks involved in trying to find a like-minded fellow student, one can post an unambiguous request for solutions anonymously, with the only risk associated being possible detection in the submitted work. Indeed, the anonymous nature of these sites lends itself to open discussion of newer and potentially more innovative methods of cheating and of preventing or identifying cheating. For example, two threads found on the site reddit.com document a student's attempt to post incorrect exam solutions to entice those using these sites to submit poor exams, thus boosting the score of the architect of this system, helping his score further by "manipulating the curve" [10]. Another thread documents a professor who "created his own chegg account and answered the question with a bullshit solution that seems right at first glance but is actually fundamentally flawed", eventually finding that 14 of the 99 students in class submitted this incorrect solution [11]. As one might expect. This also means an increase in students openly creating and posting resources in order help students more effectively cheat by avoiding detection, such as "How to Cheat in CS 101" [12].

Even in-network dishonesty has changed, as students can now share folders over a network and effectively share workspaces on tests that require the use of computational resources. While the focus of this work will be examining these issues through the lens of coding assignments, similar online resources were found for sketching assignments, essays and 3D modeling assignments.

Our course:

This large mid-Atlantic university has a general engineering program. Students are admitted with a major of General Engineering, and select a disciplinary major toward the end of their first year of study. Almost all incoming engineering students take a two-course sequence of 2-credit Foundations of Engineering courses. The initial course has no prerequisites, while the prerequisite for the second course is the first course. The course objectives for the first course (ENGE 1215) are:

Foundations of Engineering (1): As a result of this course a student will be able to:

Compare and contrast the contributions of different types of engineers in the development of a product, process, or system.

1. Develop a plan of study for your undergraduate career
2. Articulate holistic issues that impact engineering solutions
3. Solve problems using systematic engineering approaches and tools
4. Model an engineering system
5. Synthesize information from several sources
6. Communicate information effectively
7. Contribute effectively to an engineering team

The second course is a project-based course. Student teams are formed, and each section has a specified project. Student teams progress through an engineering design process to design and prototype a device according to their section. Foundations of Engineering (2) (ENGE 1216) course objectives are as follows:

Foundations of Engineering (2): As a result of this course a student will be able to:

1. Demonstrate the ability to use various engineering tools in solving design problems, including MATLAB, Inventor, and physical prototyping
2. Demonstrate proficiency with implementing an engineering design process, a. Collect, analyze, represent, and interpret data
    a. Use systematic methods to develop solutions for problems
    b. Identify all relevant stakeholders, constraints, and needs
3. Communicate engineering decisions to technical managers,
4. Contribute effectively to an engineering team.
5. Evaluate ethical implications of engineering solutions

Both courses were offered in sections of no more than 32 students. In the 2018-2019 academic year, this meant that 62 sections of 1216 (the course in question) were offered for approximately 1861 students. Notably, the Honor Code language is included in the syllabus for each section per university directive.

ENGE 1216 is broken into two main projects. Track 1 was the design and development of a drone, which is eventually tested in a drone cage with a motor and propeller. Track 2 was the design of a wind turbine designed to generate a minimal amount of power. Both projects had a set of common assignments including MATLAB assignments.

MATLAB was introduced as a topic in 1215 consistently during the final third of the semester. As a result, the “distance” students had from the subject varied based on their track in 1216, which allowed differentiation between the two courses based on the date of the infractions. Track 1 classes [drone] began working with MATLAB in the first or second week of the semester, while track 2 [turbine] classes didn’t begin MATLAB until the middle of the semester. While there was some variation in assignment content and details between the two tracks, a number of assignments, including the “Trajectory” assignment, were essentially identical between the two. Regardless of track, students are expected (and encouraged) to collaborate on assignments; this is not treated as an honor code violation. In addition, due to the nature of the assignments, a certain degree of similarity is expected, especially among students who discuss the code with each other (also encouraged, and not an honor code violation). Collaboration (as opposed to copying) is a topic discussed in instructional faculty meetings.

MATLAB, previously introduced in 1215, was a major focus of 1216 and had approximately five associated homework assignments, most requiring the use of code syntax and structure introduced in accompanying lectures. MATLAB was also used as a predictive design and analysis element in both projects, and the results presented as part of the final reports and presentations. In each individual assignment, students were provided with written prompts and specific goals and formatting requirements for the requested code. In some assignments, flowcharts or sample outputs were provided in order to provide students additional guidelines for the desired results.

The trajectory assignment:

The “Trajectory” assignment was a programming assignment assigned to both class sections at different times. For track 1 classes, the assignment was assigned around February 2nd, 2018 and due approximately February 15th, 2018, depending on exact class schedule. For the track 2 classes, it was assigned in mid-March (around Spring Break) and due approximately April 4th, 2018. Given the known level of communication between the two tracks (students often maintain social bonds formed in the first semester, and were unable to choose which track to enter), it is assumed that track 2 students were aware of the existence of the trajectory assignment long before being provided it as a result of normal social interaction.

In the assignment, students were provided with the relevant kinematic equations that would model a simplified ballistic trajectory, including worked-out calculations for horizontal and vertical distance traveled, given an initial velocity. Using the given equations, students were asked to develop a flowchart and accompanying MATLAB script that would:

1. simulate the trajectory of a projectile as a function of time,
2. output the values for time, x and y positions, x and y velocities and total velocity in a formatted table (the intent being for students to use the fprintf function in MATLAB) to a .txt file, and
3. determine maximum *simulated* elevation of said projectile,

given the following user inputs:

1. units used for calculations,
2. the corresponding value of gravitational acceleration,
3. initial launch angle in degrees,
4. initial velocity (in units defined earlier),
5. the time increment at which the simulation was to be run.

The assignment was provided to students as the culmination of the lectures introducing “for” loops and the fprintf function. While students were not necessarily required to use fprintf to produce their formatted tables, the use of “for” loops was required. This, paired with the specific formatting requested for the output ‘table’ and the flexibility of the structure internal to the programming loop, made submissions predictable in some aspects, and highly variable in others. For example, the output section of the code was nearly uniform throughout submissions, but the order and form of calculations used for the simulation could vary wildly from student to student. This contrasts with many other assignments in the course, in which many assignment code was so simple as to be impossible to solve in more than one or two ways. As a result, outside of some grievous error (e.g. submitting an assignment with another student’s name on it, which did occur), other similarly complex assignments were far less amenable to analysis via MoSS by comparison, despite a comparable level of complexity.

This assignment was one of the more complex assignments given during the semester, and required the use of loops, proper formatting, and functions. Effectively using functions required a high degree of understanding, both on the part of the students and the faculty teaching the content. We suspect that these factors, in addition to the apparent complexity of the code (as it involved physics equations), are key elements in this being the assignment with the most cases of academic misconduct throughout the semester. This assignment also had the greatest degree of variation in the types of potential academic misconduct in the semester, with violations appearing in multiple different forms, from multiple different groups of students. For example, we found cases where students had allegedly copied one-on-one, ‘borrowed’ another student’s laptop, shared student-created files within a group, downloaded solutions from Chegg.com, etc. One large group of students included the use of a somewhat obscure, irrelevant, and advanced function “uint64” in their MATLAB code, which in this case, served no purpose (and could arguably be counter to data manipulation methods discussed in class), and was not a concept or function that had been introduced in the course.

Due to the number of times this unusual function appeared, the assignments were further examined and additional evidence of infractions were discovered. Similar events occurred with typos in the comments or variables (for example, “horizentol velocity” instead of “horizontal velocity”, or “lunch angle” instead of “launch angle” appearing in identical locations through the code). It was the discovery of this large number of previously undetected violations that led to an examination of the course submissions via MoSS in a broader sense.

Institutional Honor Court:

The Office of Undergraduate Academic Integrity is quite visible at the large mid-Atlantic univerity. All courses are instructed to include the honor code in the syllabi, and students see presentations on the honor code during orientation. Faculty workshops are readily available. The office maintains a website with definitions of academic misconduct and procedures for faculty and students to follow if they suspect academic misconduct. The procedure typically involves the student’s appearance in front of an honor court, and the recommended penalty is an F*: and F tagged for academic misconduct (although less severe penalties are the norm).

MoSS:

MoSS (Measure of Software Similarity), first developed in 1994, is “an automatic system for determining the similarity of programs” [3], created by Alex Aiken in order to identify code samples that have significant similarity in structure or content. MoSS uses an algorithmic approach [13] that provides the user a percentile degree of similarity between any two programs.

For example, a pair of programs might receive a 60% and a 90% score, respectively when compared to each other--that would indicate that MoSS' analysis indicated that 60% of the first program's code was similar to the second and 90% of the second program's code was similar to the first. Notably, MATLAB makes it exceptionally easy to change variable names, offering to change every instance of a variable when the user changes one variable.

In its current form, a set of files to be examined (containing source code) are submitted to a Stanford server [3] on which MoSS is running, the analysis is performed, and results hosted for a short period of time. MoSS is capable of processing multiple programming languages (MATLAB, C, java and more), and can be instructed to ignore code structures that has appeared at least N times in its analysis (for example, if an instructor provided scaffolded code, the scaffolding would be automatically ignored). While simple codes are often identified as having a very high degree of similarity to each other, due to the system's ability to ignore elements that appear repeatedly in a sample set (e.g. example structures provided in class), the relatively extreme cases, such as copied code, still receive proportionally higher scores.

One potential downside to the approach used by MoSS appears when there are an *extremely* high number of very similar codes submitted; when a sufficient number of plagiarized files are submitted, MoSS may begin to treat copied elements as ignorable code, making it more difficult to parse out copying from code that is similar due to convergence of styles or lecture content. This being said, while it does make working through the results more difficult, as the confirmed cases were filtered out and removed from the submitted samples, the winnowed results began to conform to a more reasonable distribution of similarities.

Online sites:

There are currently a variety of "academic assistance" sites, ranging from "tutoring" websites that purport to provide students with one on one assistance, to sites designed to share resources submitted by students. Examples include Chegg.com to CourseHero.com, where students upload their course documents for general access. In both of these cases, students pay a monthly fee for full access to the resources. One popular application of resources like Chegg is not to request general assistance, but to ask direct questions and receive solutions, with students often uploading original copies of the assignment and requesting a complete solution as their "help". In the case of Chegg, the "tutors" that solve these problems are paid $20 USD per hour, billable to the minute, paid via paypal. As a result, some "tutors" have logged incredibly high numbers of provided solutions; one example was a single user that had provided over 8,000 contributions in two years. Consequently, there is significant incentive to provide solutions and receive positive feedback, rather than the extensive, sometimes painful process of one-on-one tutoring. Given a choice between tutoring and a solution to a specific problem, students certainly would opt for the solution.

With respect to CourseHero.com, while there is no direct aid being provided, the presence of prompts, lecture notes, tests, solutions and so forth may serve as an augmented version of the resource-sharing between students that already exists, but with years of material being collected in one centralized location. Students also appear to recognize that this is a potential violation of the honor code, as class documents are often listed filed under erroneous classes, or in clearly pseudonymous class names that still make reference to the original class without being casually searchable. However, one major novel risk posed by CourseHero.com is that students rarely, if ever, redact personal information from their documents, sometimes including confidential university identification numbers. Many are even visible in the free previews, as freshman students often erroneously include them on their assignments. While beyond the scope of this work, this information could easily be used for identity theft, as it would provide a potential attacker willing to proceed through a few public record checks, all the information necessary to gain access to confidential university records.

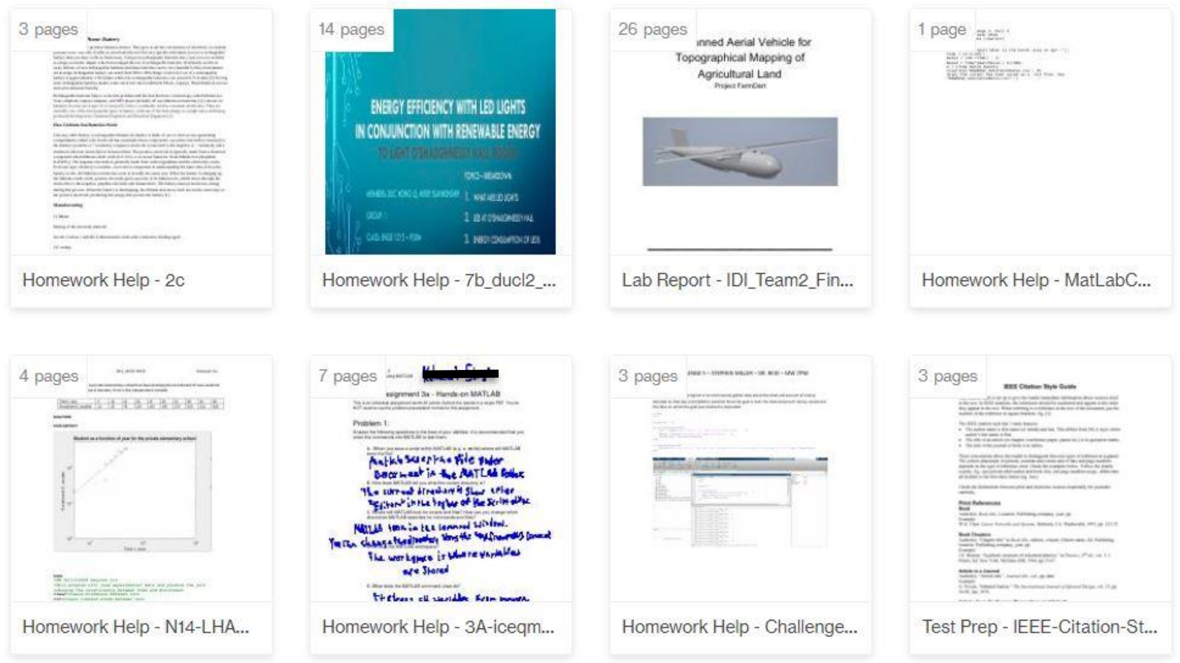
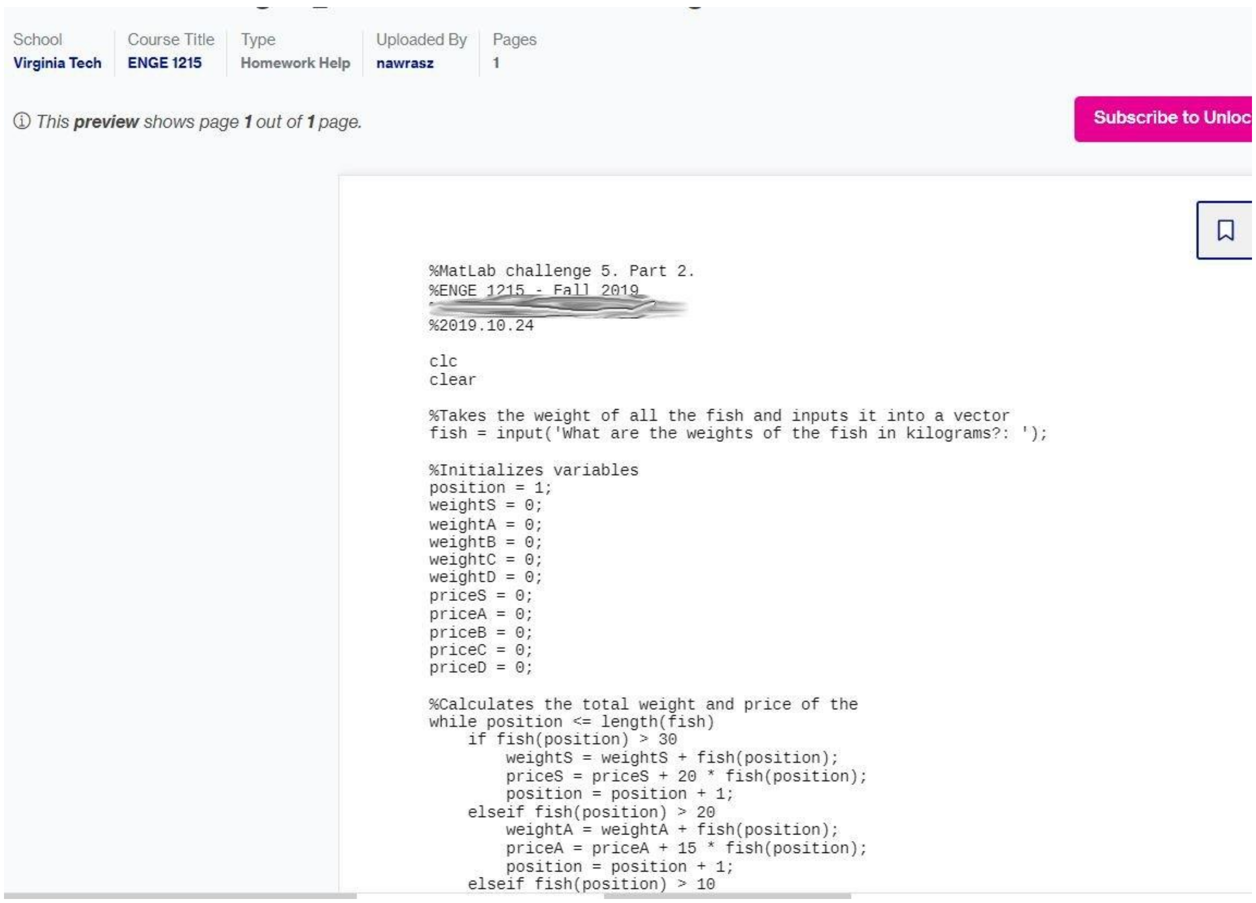

3 pages
Homework Help - 2c
14 pages
Homework Help - 7b_ducl2_...
26 pages
Lab Report - IDI_Team2_Fin...
1 page
Homework Help - MatLabC...
4 pages
Homework Help - N14-LHA...
7 pages
Homework Help - 3A-iceqm...
3 pages
Homework Help - Challenge...
3 pages
Test Prep - IEEE-Citation-St...

Figure 1: Multiple documents available in Coursehero.com when searching for ENGE 1215. Search of less than 5 clicks, no subscription. Student name intentionally hidden.


School Virginia Tech | Course Title ENGE 1215 | Type Homework Help | Uploaded By nawrasz | Pages 1

This **preview** shows page **1** out of **1** page.

Subscribe to Unloc

```
%MatLab challenge 5. Part 2.
%ENGE 1215 - Fall 2019
[redacted]
%2019.10.24

clc
clear

%Takes the weight of all the fish and inputs it into a vector
fish = input('What are the weights of the fish in kilograms?: ');

%Initializes variables
position = 1;
weightS = 0;
weightA = 0;
weightB = 0;
weightC = 0;
weightD = 0;
priceS = 0;
priceA = 0;
priceB = 0;
priceC = 0;
priceD = 0;

%Calculates the total weight and price of the
while position <= length(fish)
    if fish(position) > 30
        weightS = weightS + fish(position);
        priceS = priceS + 20 * fish(position);
        position = position + 1;
    elseif fish(position) > 20
        weightA = weightA + fish(position);
        priceA = priceA + 15 * fish(position);
        position = position + 1;
    elseif fish(position) > 10
```

Figure 2: Sample solution available at Coursehero.com for ENGE 1215. Student name intentionally blurred for this publication. Available without subscription.

Method:

All "Trajectory" assignment submissions for all sections of the course were uploaded to the MoSS system and the analysis was run. Normally, at this point, one to two members of the teaching team would review most of the resulting comparisons. Based on one author's experience, in a "standard" MoSS run for the course, matching code percentages below 50% were almost exclusively circumstantial or obviously independent work and would be ignored, while matches above 50% would be examined, at least cursorily, starting from the highest match percentages and working down. Unexpectedly, a large number of submitted assignments were either near identical, or actually identical in content, including spaces in between lines and characters, which are not relevant in MATLAB and often serve as a fingerprint of the author. Consequently, due to the large number of high-similarity submissions, assignments that would have normally been flagged for a high degree of matching code were flagged as far lower than normal; in other words, so many assignments were sufficiently similar that the MoSS algorithm assumed the matching code must have been given. After manually reviewing a number of assignments which were fundamentally identical, and using a Chegg.com account to obtain samples of posted solutions to the assignment, a Python program was written to filter out any cases that were either Chegg.com-derived, or had features unique to the unusual, but identical assignments submitted. Once these submissions had been identified and removed, the remainder of the submissions were re-uploaded for examination. Two members of the teaching team reviewed each match that MoSS flagged, and evaluated the codes side-by-side. The characteristics under review included, but weren't limited to:

- Shared unusual typos, or misspellings of words
- Shared use of functions that either had not been introduced in class or served no purpose in the code
- Student names that were other than the student submitting the code
- Identical whitespace throughout the body of the code
- All operations occurring in the same order and data being stored in the same fashion

Again, replacing variables in MATLAB is a relatively simple process, involving the use of a "find-and-replace" GUI window that allows for simultaneous replacements of code and/or whitespace throughout the document. As a result, similarity (or not) of variables was rarely if ever the only deciding factor. In many cases, even several of these elements matching between two codes was insufficient to be *certain* that there had been academic misconduct, and were consequently rejected for submission to the honor court. However, many cases had at least three or more characteristics, and passed the 'window test'; imagine if printouts of the code were aligned and held up in front of a window - looking through one to the other shows that the characters were identical. In other words, codes suspected of violation were often so similar that they could have been overlain on one another flawlessly, and had perfect alignment, despite variable changes, or the addition of comments. Once we had established beyond any reasonable doubt that the codes were too similar to have been created by legitimate collaboration (i.e. two or more students discussing the code, but writing their own versions), the cases were added to a spreadsheet that had been built and were submitted to the honor court. There were also several incidents of students clearly trying to "muddy the water" by changing their variables consistently throughout their submissions, which due to the nature of MoSS, was ineffective. One particularly memorable incident involved one student using the variables "A", "B", "C", "D" and "E", while their counterpart used "F", "G", "H", "I" and "J", without making any other changes to the code. The very nature of the variable names and exact avoidance made identification far simpler than in other instances.

In cases of disagreement or where the reviewers felt a case may be flawed, the code was not submitted: in all of these cases, there was potentially insufficient proof of academic misconduct. This process, despite being mechanically simple, required several weeks of meetings between the reviewers, and the temporary hiring of a member of the university just to sort through the results. The entire process took several months in total and was unexpectedly time consuming, and expensive.

Results:

Overall, we can review the total number of cases and their characteristics:

Table 1: Count of student infractions

Student pleas:	number	percent
Pled Guilty:	147	78.19%
Asked for meeting:	8	4.26%
Pled Not-Guilty:	17	9.04%
No Response:	15	7.98%
Guilty, but didn't accept sanction:	1	0.53%
Total students:	**188**	

Incidents (possibly involving multiple students)	number	
Incidents involving more than one instructor	22	
Incidents involving one instructor	29	

Table 2: Count of student infractions, trajectory assignment only

Trajectory only, student cases:	number	Percentages compared to *all* infractions
Chegg only cases/students (Trajectory):	20	10.6%
Traditional MoSS students (Trajectory):	74	39.34%
Total students:	**94**	50.0%

Incidents (possibly involving multiple students)	number	
Trajectory Incidents (Track 1 - only):	18	35.29%
Trajectory Incidents (Track 2 - only):	25	49.02%
Mixed Class Incidents (1 >> 2):	8	15.69%

Tables 1 and 2 show the number of individual cases and number of incidents for all cases and for the trajectory assignment. Table 1 shows a total number of 188 students and their response to learning that they were turned into the honor court. Incidents record a number of unique incidents. For example, in one case, multiple students spread among four different instructors were found to have submitted identical code; this would count as 1 incident.

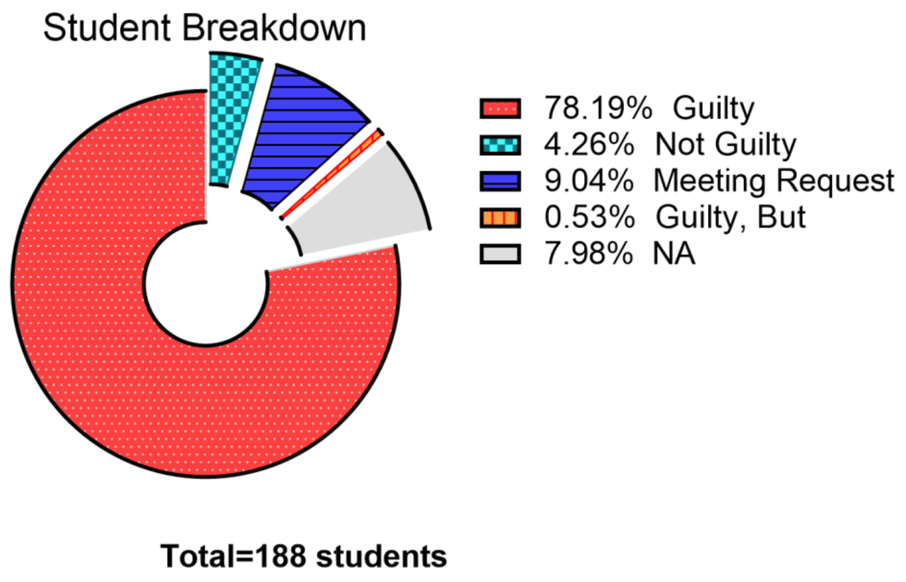
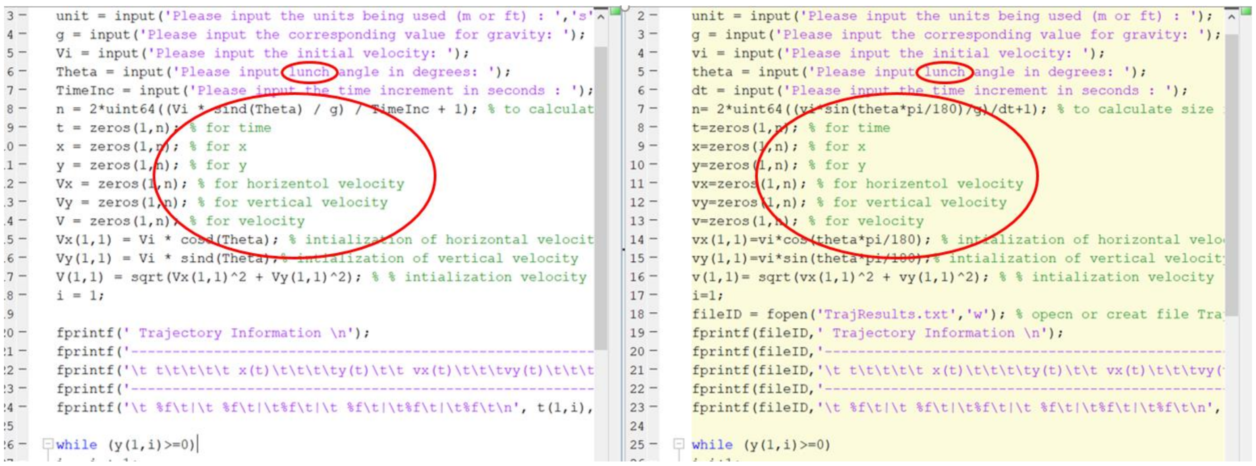


Figure 3: Visual representation of student guilty vs. not guilty pleas

Table 2 shows statistics for the trajectory assignment, we find 20 cases of students submitting Chegg solutions and 74 cases of copying without obvious Chegg involvement. Upon inspection, there were only 38 total students operating either as a singlet (using Chegg) or in a pair (with one instructor). The remaining 56 students were all cross-instructor, or cheating among different sections.

The incidents in Table 2 illustrate 8 cases of copied code from one project group to the other, clearly showing a transfer of the solution from one group (who used MATLAB weeks prior) to the other.

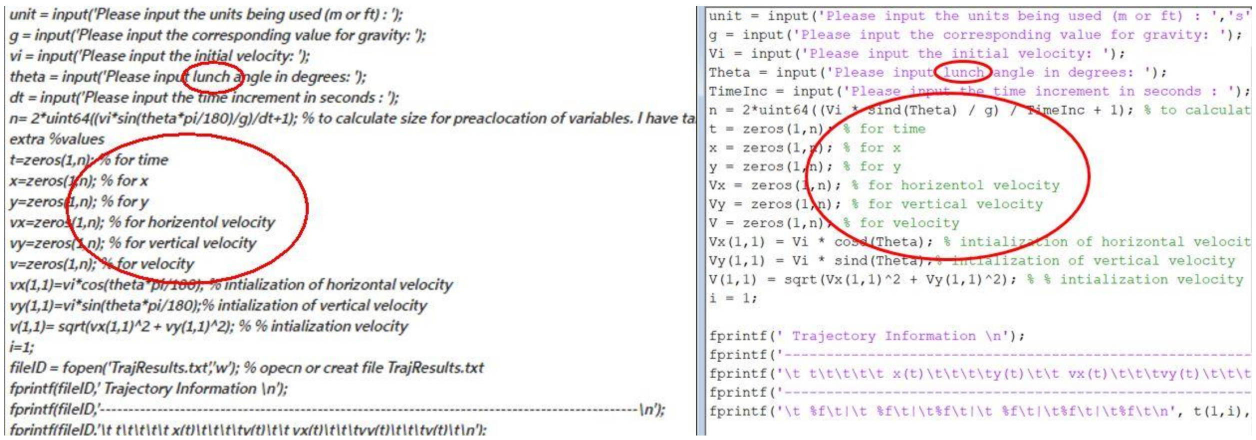
We found that the majority (64.71%) of cases were related to the track 2 class sections, with 49.02% of cases involving only track 2 students and 15.69% involving both track 1 and track 2 students. However, given that track 1 students submitted their work far in advance of track 2 students, we can treat this as "flow" from track 1 to track 2. In other words, it is likely that a student in the track 1 group shared code with a student in the track 2 group given the timing of the assignments. This is reflected primarily in the shared work unique to students, and not Chegg.com-tied submissions. Of the twenty Chegg.com-tied submissions, nine were related to track 2 students and eleven to track 1 students, implying that there was not an increased use in Chegg.com throughout the semester, and that the increase in cases was due to other factors. This could also indicate that the known availability of solutions in the students' peer group was the primary factor, and that students who used Chegg.com were already aware of it (and likely using it) prior to the assignment. The bulk of the cases were pairs of students, with an average incident size of 2.38 students. Based on this, while cross-track sharing did seem to increase over the semester, nearly 40% of the incidents on this assignment were derived, directly or indirectly, from Chegg.com.



```
unit = input('Please input the units being used (m or ft) : ','s'
g = input('Please input the corresponding value for gravity: ');
Vi = input('Please input the initial velocity: ');
Theta = input('Please input lunch angle in degrees: ');
TimeInc = input('Please input the time increment in seconds : ');
n = 2*uint64((Vi * sind(Theta) / g) / TimeInc + 1); % to calculat
t = zeros(1,n); % for time
x = zeros(1,n); % for x
y = zeros(1,n); % for y
Vx = zeros(1,n); % for horizentol velocity
Vy = zeros(1,n); % for vertical velocity
V = zeros(1,n); % for velocity
Vx(1,1) = Vi * cosd(Theta); % intialization of horizontal velocit
Vy(1,1) = Vi * sind(Theta);% intialization of vertical velocity
V(1,1) = sqrt(Vx(1,1)^2 + Vy(1,1)^2); % % intialization velocity
i = 1;

fprintf(' Trajectory Information \n');
fprintf('-----------------------------------------------------
fprintf('\t t\t\t\t\t x(t)\t\t\t\ty(t)\t\t vx(t)\t\t\tvy(t)\t\t\t
fprintf('-----------------------------------------------------
fprintf('\t %f\t|\t %f\t|\t%f\t|\t %f\t|\t%f\t|\t%f\t\n', t(1,i),

while (y(1,i)>=0)
```

```
unit = input('Please input the units being used (m or ft) : ');
g = input('Please input the corresponding value for gravity: ');
vi = input('Please input the initial velocity: ');
theta = input('Please input lunch angle in degrees: ');
dt = input('Please input the time increment in seconds : ');
n= 2*uint64((vi*sin(theta*pi/180)/g)/dt+1); % to calculate size
t=zeros(1,n); % for time
x=zeros(1,n); % for x
y=zeros(1,n); % for y
vx=zeros(1,n); % for horizentol velocity
vy=zeros(1,n); % for vertical velocity
v=zeros(1,n); % for velocity
vx(1,1)=vi*cos(theta*pi/180); % intialization of horizontal velo
vy(1,1)=vi*sin(theta*pi/180);% intialization of vertical velocit
v(1,1)= sqrt(vx(1,1)^2 + vy(1,1)^2); % % intialization velocity
i=1;
fileID = fopen('TrajResults.txt','w'); % opecn or creat file Tra
fprintf(fileID,' Trajectory Information \n');
fprintf(fileID,'------------------------------------------------
fprintf(fileID,'\t t\t\t\t\t x(t)\t\t\t\ty(t)\t\t vx(t)\t\t\tvy(
fprintf(fileID,'------------------------------------------------
fprintf(fileID,'\t %f\t|\t %f\t|\t%f\t|\t %f\t|\t%f\t|\t%f\t\n',

while (y(1,i)>=0)
```

Figure 4: Sample MATLAB code submitted by two individuals. Note that variable names are changed (typical when trying to avoid detection by the system).



```
unit = input('Please input the units being used (m or ft) : ');
g = input('Please input the corresponding value for gravity: ');
vi = input('Please input the initial velocity: ');
theta = input('Please input lunch angle in degrees: ');
dt = input('Please input the time increment in seconds : ');
n= 2*uint64((vi*sin(theta*pi/180)/g)/dt+1); % to calculate size for preaclocation of variables. I have ta
extra %values
t=zeros(1,n); % for time
x=zeros(1,n); % for x
y=zeros(1,n); % for y
vx=zeros(1,n); % for horizentol velocity
vy=zeros(1,n); % for vertical velocity
v=zeros(1,n); % for velocity
vx(1,1)=vi*cos(theta*pi/180); % intialization of horizontal velocity
vy(1,1)=vi*sin(theta*pi/180);% intialization of vertical velocity
v(1,1)= sqrt(vx(1,1)^2 + vy(1,1)^2); % % intialization velocity
i=1;
fileID = fopen('TrajResults.txt','w'); % opecn or creat file TrajResults.txt
fprintf(fileID,' Trajectory Information \n');
fprintf(fileID,'--------------------------------------------------------------------------------------\n');
fprintf(fileID,'\t t\t\t\t\t x(t)\t\t\t\ty(t)\t\t vx(t)\t\t\tvy(t)\t\t\tv(t)\t\n');
```

```
unit = input('Please input the units being used (m or ft) : ','s'
g = input('Please input the corresponding value for gravity: ');
Vi = input('Please input the initial velocity: ');
Theta = input('Please input lunch angle in degrees: ');
TimeInc = input('Please input the time increment in seconds : ');
n = 2*uint64((Vi * sind(Theta) / g) / TimeInc + 1); % to calculat
t = zeros(1,n); % for time
x = zeros(1,n); % for x
y = zeros(1,n); % for y
Vx = zeros(1,n); % for horizentol velocity
Vy = zeros(1,n); % for vertical velocity
V = zeros(1,n); % for velocity
Vx(1,1) = Vi * cosd(Theta); % intialization of horizontal velocit
Vy(1,1) = Vi * sind(Theta);% intialization of vertical velocity
V(1,1) = sqrt(Vx(1,1)^2 + Vy(1,1)^2); % % intialization velocity
i = 1;

fprintf(' Trajectory Information \n');
fprintf('--------------------------------------------------
fprintf('\t t\t\t\t\t x(t)\t\t\t\ty(t)\t\t vx(t)\t\t\tvy(t)\t\t\t
fprintf('--------------------------------------------------
fprintf('\t %f\t|\t %f\t|\t%f\t|\t %f\t|\t%f\t|\t%f\t\n', t(1,i),
```

Figure 5: Sample code (student submission) shown with code posted on Chegg.com

Discussion:

Overall, prominent copying has been observed between class sections, instructors, and even class tracks or project groups. An optimistic view would be that the course is successfully creating lasting bonds in the first semester, and the social networks are maintaining cohesion despite students being split up in subsequent classes. However, we also see that the ratio of “network” cheating, in which students collaborate with each other, is almost equal to that of individual cheating using anonymous online resources. Students are also aware of the tools being used, both by faculty and other students and, anecdotally, are capable of manipulating both. For example, the available github archive, titled “how to cheat in computer science 101” directly addresses MoSS as a detection tool and provides instructions for how to evading detection. In these cases, course grades are being indirectly or directly manipulated by students with the aid (or interference) of other students. This begs the question of how accurate the evaluations are, when student manipulation of resources (illicit or otherwise) is a very real possibility, given the plethora of online (and anonymous) resources.

In the past, academic dishonesty has relied heavily on physical and social peer networks. Outside of bringing in illicit materials, you often had to know “the right person”. This localized the incidents and restricted their spread. The newer methods we have discussed here are social network agnostic, anonymous in nature, and fundamentally lower risk, as peers, who may or may not agree with the behavior, are not involved. An added rationale for the behavior is that by posting material or solutions online, a student can justify any perceived indiscretion and feel like they are helping their peers; given the lack of personal involvement and interaction, this illusion can be maintained.

Even within “in-network” incidents, a minimum of 43% of the incidents involved students in separate class sections, as different instructors were involved; of the 94 students involved in the “Trajectory Assignment” cases, 56 of them collaborated across class-lines. From this it seems clear that copying was far from limited to close friends in the same class section and while this may speak to the successful team building in the first semester’s 1215 course, and creating a stable community with strong connections, this is obviously far from the desired behavior.

The academic dishonesty observed was also not the result of an immediate risk of failure within the class; in the case of the “Trajectory assignment” the assignment itself was worth less than 1.75% of the final grade, and was assigned at a point in the semester long before final grades would begin to solidify. Given that a number of students were also repeat offenders, with one having infractions on all but one MATLAB assignment, this seems to indicate a pattern of behavior, rather than an exception.

As is often the case, some faculty were reluctant to examine the problem or submit cases to the honor court. Objections from individuals ranged from a sense of being unfair to students. “Did we tell the students we would submit their code to online resources?” In fact, we had, as this was in the syllabus. Some individuals still felt that students were owed an immediate notification each time they were submitted. Other individuals expressed a concern as the cases were being submitted very near the end-of-semester faculty evaluation, and faculty were apprehensive about any ramifications.

Given these issues, what can be done?

Many institutions have an office of academic integrity, or policies in place designed to submit suspected cases. However, the enforcement eventually relies on individual faculty members to track academic misconduct; this is, at best, inconsistent. There are departments inside and outside of engineering that ignore the issue: exams and homework assignments are reused year after years. Crafting individual assignments or incorporating a reflective element may help reduce academic dishonesty. Tracking and processing cases may require a dedicated person within a department or college (as was the case here). In this case, this person may need to subscribe to Chegg, CourseHero, and similar sites.

Conclusion:

Due to the methodology used, and the standards set during the analysis, there is little to no chance that a student with an acceptable understanding of code would be detected using our methods. For example, a student who understood the nature of whitespace, and that varying whitespace would change the appearance of code but not the functionality, could (and possibly did) easily evade detection. As a result, it is likely that these numbers represent only the most desperate, most time-limited, or least capable students in the course. One pessimistic view of our finding is: 10% of a large class demonstrated such a lack of understanding that they submitted code that they could not (or would not attempt to) understand.

This represents one instance at one university in one class in one semester. If we consider that cases with *any* doubt were assumed to be innocent (and not investigated any further) and the number of cases was a decent percentage of all incoming engineering students, we can conclude: cheating is still prominent (no surprise), and the methods employed are advancing. Additionally, students are aware of not only these resources, but faculty's awareness of those resources as well, and often make adjustments to deliberately evade detection resources, or to conceal the source of their work. In this course, prior to an organized effort including comparison between sections and instructors, we assumed the former methods of cheating were prominent: "keep your eyes on your own paper" and emailing of solutions among friends in a single section of a class. After this case, the results suggest that, due to the changes in social networks and available tools, the traditional methodology for cheating prevention is no longer effective. With a second or two of access to a cellphone, a student can (and will) upload a question and receive an answer, and materials from the past semester to the past decade are routinely mass-uploaded to the internet and archived permanently. Current tools, including the reporting methods used by the university, are extremely time consuming or not structured for the high case numbers involved. Anecdotally, other departments have faced similar issues, both in detection and reporting, almost all in courses involving significant coding components. As a result, improving cheating prevention and detection may require the development of tools and techniques more closely related to computer forensics and cybersecurity than traditional methods. Otherwise, in an increasingly connected world of available information, the source of that information may become increasingly difficult to verify.

In the end, anecdotally, we can still observe that a student with an acceptable level of knowledge suitable to avoid detection has every ability to simply write the program – in other words, with *comparable* effort, most students could simply learn the material.

References:

1. Reid, K.J., Reeping, D. & Spingola, E.: "A Taxonomy for Introduction to Engineering Courses," ***International Journal of Engineering Education***, Vol. 35, No. 1, 2018.
2. Honor Code Policy and Manual, URL: https://honorsystem.vt.edu/honor_code_policy_test.html, accessed 2/1/2020
3. A Theory for Detecting Software Similarity, URL: https://theory.stanford.edu/~aiken/moss/, accessed 2/1/2020.
4. Roth, N.L. and McCabe, D.L., "Communication Strategies for Addressing Academic Dishonesty," J. College Student Development, vol. 36, n. 6, 1995, pp. 531-541.
5. McCabe, D.L. and Makowski, A.L., "Resolving Allegations of Academic Dishonesty," About Campus, March-April, 2001, pp. 17-21
6. Carpenter, D., Harding, T., Finelli, C., Montgomery, S., & Passow, H., "Engineering Students' Perception of and Attitudes Toward Cheating," Journal of Engineering Education, July 2006.
7. Kibler, W.L., "Academic Dishonesty: A student development dilemma," NASPA Journal, vol. 30, no. 4, 1993, pp. 252-257.

8. Harding, T.S., Carpenter, D.D., Montgomery, S. & Steneck, N.H. “A comparison of the role of academic dishonesty policies of several colleges on the cheating behavior of engineering and pre-engineering students,” Frontiers in Engineering Conference Proceedings, 2002.
9. Harding, T.S., Carpenter, D.D., Montgomery, S. & Steneck, N.H. “The current state of research on academic dishonesty among engineering students,” Frontiers in Engineering Conference Proceedings, 2001.
10. “AITA for intentionally posting the wrong test answers on a cheating website?”, URL: https://www.reddit.com/r/AmItheAsshole/comments/dyi6m8/aita_for_intentionally_posting_the_wrong_test/, accessed 2/1/2020.
11. “Not my revenge, but my professor against cheater”, URL: https://www.reddit.com/r/ProRevenge/comments/e8f0z1/not_my_revenge_but_my_professor_against_cheaters/, accessed 2/1/2020
12. “How to Cheat in CS 101”, URL: https://github.com/genchang1234/How-to-cheat-in-computer-science-101, accessed 2/1/2020
13. Schleimer, S., Wilkerson, D., & Aiken, A., “Winnowing: Local Algorithms for Document Fingerprinting,” SIGMOD Conference Proceedings, 2003.