Active Academic Integrity

Alex Kassil

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-157
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-157.html

May 21, 2021

Acknowledgement

I would like to thank Michael Ball for trusting me, mentoring me, and always being there for me when I needed anything! It has been a fun couple of years. Michael's support opened up a world of Computer Science Education research to me. I would like to thank John DeNero teaching me, working with me, and supporting me! I have learned so many wonderful things from John. A special thank you to Catherine Cang for all her support. Finally a huge shoutout to my parents Irina Kassil and Victor Fedotov for their unwavering love.

Active Academic Integrity

by Alex Kassil

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor John DeNero
Research Advisor

May 11, 2021

(Date)

* * * * * * *

Lecturer Michael Ball
Second Reader

May 12, 2021

(Date)

UNIVERSITY OF CALIFORNIA, BERKELEY

Active Academic Integrity

by

Alex Kassil

A thesis submitted in partial fulfillment for the
degree of Master of Science

in the
Department of Electrical Engineering and Computer Sciences

May 21th, 2021

UNIVERSITY OF CALIFORNIA, BERKELEY

Abstract

Department of Electrical Engineering and Computer Sciences

Master of Science

by Alex Kassil

The response to academic misconduct in introductory computer science courses can potentially affect the behavior and culture of students throughout their time as undergraduates. Moss [1], the most commonly used system for detecting excessive collaboration, is not well suited to assignments whose solutions are only a few lines long. This paper describes an active approach to detecting the use of solutions from prior semesters in which instructors change the names of various identifiers, change constants, and change logic used in the problem. We developed a tool that searches student submission history to detect the use of identifiers, constants, and logic from prior semesters. As with TMOSS [2], the misconduct is detected across the entire history of partially completed student work, not just their final submission. Our developed tool was first used in CS 61A which had a Spring 2020 enrollment of around 1,800 students. There were 164 students notified and 142 penalties enforced (86.6% of notified students) for the first homework. Over the Spring 2020 semester, there were 452 enforced cases of misconduct on homework, labs, and projects out of 598 accusations (75.6% of cases). In Summer 2020, improvements to the tool sped up response time and reduced the false positive rate from 24.4% to 21.9%. In Fall 2020, the false positive rate was reduced to 20.5%. In Spring 2021, an automated early warning system further reduced the false positive rate to 6%. The tool has also been adapted for use by another similar course, CS 88. The paper finally discusses what was learned using this tool and communicating with hundreds of students about academic integrity, as well as best practices for instructors so students are caught breaking the rules before they develop bad habits that hinder their learning and lead to persistent cheating.

Acknowledgements

I would like to thank Michael Ball for trusting me, mentoring me, and always being there for me when I needed anything! It has been a fun couple of years. Michael's support opened up a world of Computer Science Education research to me. I would like to thank John DeNero teaching me, working with me, and supporting me! I have learned so many wonderful things from John. A special thank you to Catherine Cang for all her support. Finally a huge shoutout to my parents Irina Kassil and Victor Fedotov for their unwavering love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In college courses, including introduction to Computer Science courses, some students choose to excessively collaborate, search for answers online, or ask friends who have taken the class before to give them answers. Studies have found as many as 70.4% [4] of college students have cheated in college. Students commit academic misconduct for a variety of reasons, with some common reasons being "desire to get ahead", having the "opportunity to cheat," "cultural or moral acceptance of cheating as an established norm," "low risk of detection," or "heavy time demands" [5]. Stress and struggle are two other reasons. Whatever the reason, this misconduct takes away from the student's learning experience.

Therefore it is important to catch these students who commit academic misconduct, inform them of why they were caught, and deal with any false positives. No student wants to be falsely accused, and no instructor wants to enforce a penalty when the student did not cheat.

By utilizing staff time early in the course, our developed tool allows for quick and efficient identification of who used past semester solutions. We use this data to give students warnings, penalties, and to teach students to be honest going forward.

Chapter 2

Background and Related Work

2.1 Moss

A common system used for plagiarism detection in software is Moss [1], Measure Of Software Similarity, which automatically detects similarity between two files.

Moss finds k-grams that match between documents, where a k-gram is a contiguous substring of length k.

Previous copying detection before Moss removed irrelevant features from text, split text into many k-grams, selected a hashed subset of these k-grams to be a document's fingerprints, and checked if those hashes matched up with another document.

Here is an example:

```
Texts:
1. Alex Kassil
2. Alexander F. Kassil

Texts with irrelevant features removed:
1. alexkassil
2. alexanderfkassil

Texts split into 5-grams:
1. alexk lexka exkas xkass kassi assil
2. alexa lexan exand xande ander nderf derfk erfka rfkas fkass kassi assil
```

With a subset selection that took every other 5-gram, kassi or assil would match between the two documents and be the one fingerprint match.

There are three desirable properties of copy-detection: whitespace insensitivity, noise suppression, and position independence.

For whitespace insensitivity, everything other than text is removed, text is lowercased, and for code all variable names are swapped with "V" to achieve variable name agnostic search.

For noise suppression, a value of k needs to be selected that is not so small that the algorithm allows too many short commonalities to be found and not so long to never have matches.

For positional independence, fingerprints are chosen independent of position. One way of achieving this is choosing hashes which equal 0 mod p.

The paper on ideas behind Moss [3] describes an algorithm for selecting which hashes should be the fingerprints, with the algorithm called winnowing. The algorithm slides a window across the hashes and repeatedly selected the minimum in the window to be added to the list of fingerprints.

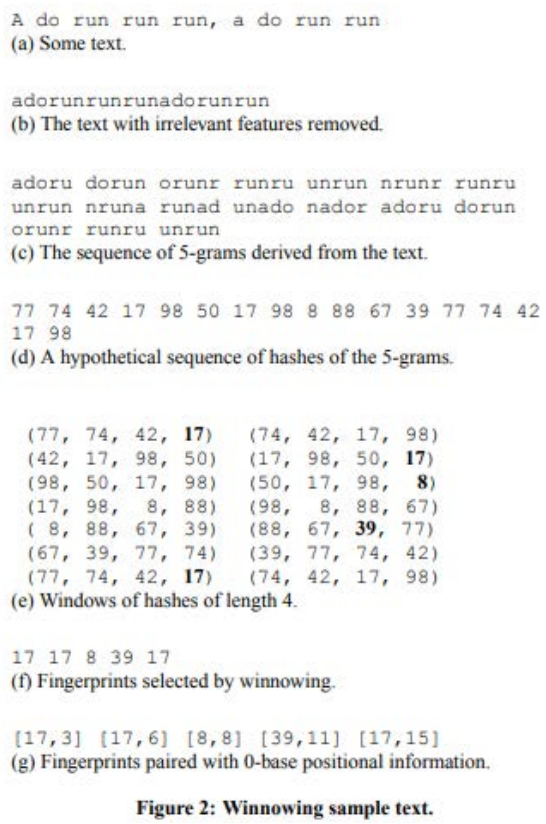

FIGURE 2.1: Winnowing example from [3]

Even after years of service, Moss performed its function well, with false positives, in the form of hash collisions, never being reported. The paper goes on to state that "users

report that copy detection does dramatically reduce the instances of plagiarism in their classes" [3].

In addition to the winnowing algorithm, another strong feature of Moss was it "being provided as an Internet service. The service has been designed to be very easy to use–you supply a list of files to compare and Moss does the rest" [1].

2.2 TAPS: A Moss Extension for Detecting Software Plagiarism at Scale

TAPS [6] stands for Tool Addressing Plagiarism at Scale. The original Moss project is effective for checking single documents for plagiarism against each other as well as against online solutions that might exist, like checking for plagiarism for one large course project. TAPS is an extension that does much of the important preprocessing when there are both multiple assignments in a class as well as multiple previous offerings of the course that have had similar or identical assignments.

The problem TAPS solves is it becomes quite cumbersome to check a current batch of assignments against each other as well as the previous ones, so TAPS:

1. Allows for mixed programming languages, and separates them before submission to Moss
2. Deals with file management, like zip archives and multiple depths of directories which need to be expanded and normalized before sent to Moss.
3. Filters, since a student should not have their nth assignment checked against their own (n - 1)th assignment, since matches are likely when assignments build upon each other.

This saved an instructor time organizing, submitting, and filtering class assignments for the purpose of software plagiarism detection by reducing the total time spent from 50 hours to only 10 minutes.

2.3 TMOSS: Using Intermediate Assignment Work to Understand Excessive Collaboration in Large Classes

An extension to Moss that is similar in spirit to our work is TMOSS [2], Temporal Measure of Software Similarity. The TMOSS algorithm relies on the fact that students often

have backups of intermediate assignment work, like git history or okpy [7] submission history. By utilizing the submission history as well as the final submission for plagiarism detection, while time analyzing increases, TMOSS is "almost twice as effective as traditional software similarity detectors in identifying the number of students who exhibit excessive collaboration" [2]. Also of interest is the paper also finds "that such students [who cheat] spend significantly less time on their assignment, use fewer class tutoring resources, and perform worse on exams than their peers" [2].

Below is the algorithm from the TMOSS paper, which adds comparison of intermediate backups of each student to every other student's submission as well as any online solutions.

ALGORITHM 1: Computing top matches in TMOSS.

```
Input: N students (students)
       n online solutions (online)
Output: N top matches (top_matches). Each student has a
        tuple of (highest similarity score, match code).
top_matches = [];
for i in 1 . . . N do
    compare_set =
        getFinalSubmissions(students - students[i]) + online;
    snapshots = getRepository(students[i]);
    student_matches = [];
    for j in 1 . . . M do
        results = compute_similarity(
            snapshots[j], compare_set);
        student_matches.append(
            argmax_{k=1...N+n-1} results[k].getScore());
    end
    top_matches.append(
        argmax_{j=1...M} student_matches[j].getScore());
end
```

FIGURE 2.2: Computing top matches in TMOSS [2]

Moss, TMOSS, and TAPS are variable name unaware. That is so a student is not able to just change variable names to avoid plagiarism detection. Active Academic Integrity on the other hand is hyper aware of variable name. Moss, TMOSS, and TAPS do not work well for programming assignments with solutions that only one or a few lines of code, like in CS 61A [8]. Active Academic Integrity does not have this problem and is complementary to using Moss/TMOSS/TAPS.

2.4 CS 61A

CS 61A: Structure and Interpretation of Computer Programs [8] is the first required Computer Science course that Computer Science, Electrical Engineering & Computer Sciences, and Data Science majors take at the University of California, Berkeley.

Week	Date	Lecture	Textbook	Orientation Links	Lab, Discussion, & Exam Prep Links	Homework & Project
1	Wed 1/20	Introduction [Rec] Lecture Q&A full 4pp 01.zip	Ch. 1.1		Lab 00: Getting Started Due Tue 1/26	
	Fri 1/22	Functions [Rec] Lecture Q&A full 4pp 02.py	Ch. 1.2 Ch. 1.3 Ch. 1.4			HW 01: Variables & Functions, Control Due Thu 1/28 Solutions
2	Mon 1/25	Control [Rec] Lecture Q&A full 4pp 03.py	Ch. 1.5	[Rec] 3pm [Rec] 3pm-NPE [Rec] 5pm [Rec] 6pm-NPE [Rec] 7pm [Rec] 10pm-NPE [Rec] Tues Lab	Lab 01: Variables & Functions, Control Due Tue 1/26 Solutions	
	Wed 1/27	Higher-Order Functions [Rec] Lecture [Rec] CombineFuncs Q&A full 4pp 04.py	Ch. 1.6	[Rec] 3pm [Rec] 3pm-NPE [Rec] 5pm [Rec] 6pm-NPE [Rec] 7pm [Rec] 10pm-NPE	Disc 01: Environment Diagrams, Control Solutions	Hog Due Fri 2/5
	Fri 1/29	Environments [Rec] Lecture Q&A full 4pp 05.py	Ch. 1.6		(Optional) Exam Prep 01: Control, Higher-Order Functions Solutions (Optional) Lost 01: Control, Environment Diagrams Solutions	HW 02: Higher-Order Functions Due Thu 2/4 Solutions

FIGURE 2.3: First two weeks of CS 61A

The course covers programming paradigms, program structures, interpreters, and methods of abstraction, and is taught in Python, Scheme, and SQL.

2.5 OK

OK is a software tool that CS 61A and many other computer science and data science courses use to collect and grade student work.

Name	Scores	Status
Lab 00	Effort: 1.0	Submitted WED 08/23 11:12 PM
Homework 1	Effort: 2.0 Total: 0.0	Submitted FRI 08/25 10:32 PM
Lab 01	Effort: 1.0 Total: 1.0	Submitted MON 08/28 04:57 PM
Hog	Revision: 2.0 Total: 19.0 Composition: 1.0	Submitted TUE 08/29 09:52 PM
Homework 2	Effort: 2.0	Submitted MON 09/04 10:44 AM
Hog Contest	–	No Submission Did you run `--submit`?
Homework 3	Effort: 2.0	Submitted FRI 09/08 12:33 PM
Lab 02	Effort: 1.0	Submitted TUE 09/05 07:51 PM

FIGURE 2.4: An example student view of OK dashboard

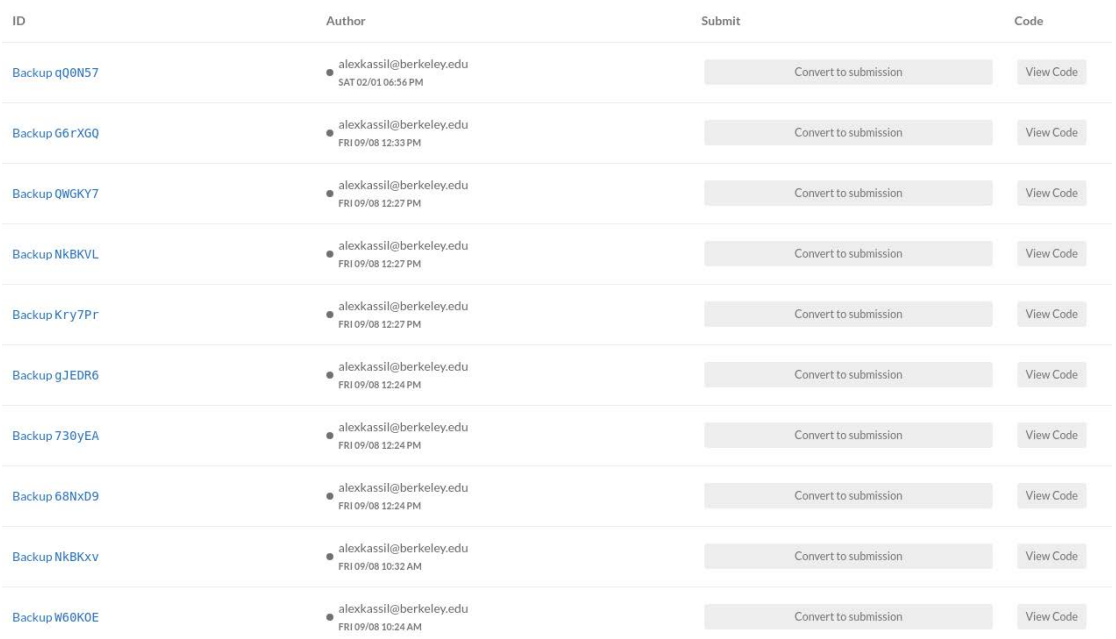

FIGURE 2.5: An example of backups for an assignment in OK

The main feature Active Academic Integrity uses from OK is the submission history. Every time a student tests their code, a snapshot of their code gets uploaded to OK. This snapshot is referred to as a backup.

Chapter 3

Overview

In Fall 2019, a problem on the first homework was changed for pedagogical reasons. When grading, a Teaching Assistant noticed that a few students failed the first homework because they simply submitted last semester's first homework.

In Spring 2020, we started actively changing variable names and program logic on the homework, lab, and project assignments. Students in this course use a system called okpy [7] to test their code locally and submit it to the autograder. Every time a student tests their code, a copy gets created as a backup submission and sent to our servers. We then download each backup for every student and search their code for past variable names or program logic. Over the course of a semester there are over a million different backups.

What we discovered is that hundreds of students at some point during the course copy code from somewhere when they are stuck on a problem. They paste the code they find online into their submission file, run the tests, then get an error message due to invalid variable names, fix their code, and then submit the file. If we just looked at their final submissions we could never tell they copied their code, but by looking at their backups we can see when the students committed academic misconduct.

Unfortunately, there were some students who committed academic misconduct, were notified, and then still committed academic misconduct on a future assignments. These students received harsher penalties on future offenses.

```
def shifty_shifts(start, goal, limit):
    """A diff function for autocorrect that determines how many letters
    in START need to be substituted to create GOAL, then adds the difference
in
    their lengths.
    """
    # BEGIN PROBLEM 6
    if start == goal:
        return 0
    if limit == 0:
        return 1
    if min(len(start), len(goal)) == 0:
        return max(len(start), len(goal))
    diff = start[0] != goal[0]
    return diff + swap_diff(start[1:], goal[1:], limit-diff)
    # END PROBLEM 6
```

```
def shifty_shifts(start, goal, limit):
    """A diff function for autocorrect that determines how many letters
    in START need to be substituted to create GOAL, then adds the difference
in
    their lengths.
    """
    # BEGIN PROBLEM 6
    if start == goal:
        return 0
    elif limit == 0:
        return 1
    if min(len(start), len(goal)) == 0:
        return max(len(start), len(goal))
    difference = start [0] != goal[0]
    return difference + shifty_shifts(start[1:], goal[1:], limit-difference)
    # END PROBLEM 6
```

FIGURE 3.1: Backup code (above) with the offending function name `swap_diff` on the second to last line, versus final submission (below) with the correct variable name `shifty_shifts`.

The figure above is an example of what the system catches. One backup, out of 97, contains `swap_diff`, the variable name used in the previous semester, as opposed to `shifty_shifts`. The final submission is a completely valid submission and contains no clue that this solution was copied.

Chapter 4

Workflow and Best Practices

This section is for other instructors to learn about how we set up our Active Academic Integrity pipeline and what we learned from over a year of academic integrity work.

The two guiding principles of our work were teaching students to be honest and minimizing student stress. These principles shaped our policies.

4.1 Setup

Both these principles influenced one of our very first assignments, the Homework 1 Syllabus Quiz. On it we had a question pertaining to academic honesty which linked to our syllabus section on academic honesty, as well as tested the students on their understanding by giving them scenarios that occurred in previous semesters. The question and answers can be found in A.

The next thing we did was download past semester course rosters and compare them to the current semester's course roster to see which students are retaking. They would also have good reason to have past semester solutions from when they took the course.

Some students work on past semester assignments during the semester. For example, to get ahead before the next project is released. We had another form that could be filled out at any time for this scenario. The question and answers can be found in B.

4.2 Assignment Changes

To be confident a student utilized a past semester solution when solving a problem, we decided to change problems. Using several semesters worth of changes, we present examples of good changes and bad changes.

4.2.1 Bad Changes

Variable name changes x to y, f to g, previous_score to prev_score are all not good. Each have the same issue. Students are likely to think of the other variable name naturally when solving the problem and not be referencing a past semester solution!

For example, if a problem was

```
def update_score(previous_score, x):
    ...
```

and the next semester the problem was

```
def update_score(prev_score, y):
    ...
```

A student might think of x as a common variable used to hold a value and use that instead of y, or might read to themselves prev_score as previous_score and use the wrong variable name. To reduce student stress, we wanted to minimize the number of false positives, and changes like these led to false positives.

Even changes like dollar to euro can lead to false positives, as seeing one currency can make a student think of another.

4.2.2 Good Changes

Good changes should make it stand out when a student copies a past semester solution.

One very effective change was for a recursive problem in a project to change the name of the function from `swap_diff` to `shifty_shifts`. The new name was themed and it was very unlikely someone would randomly type the old function name. Another was where we had a similar name for the problem, but changed what was actually asked of the student to be completely different. Backups containing valid solutions to the old version of the problem were simply nonsensical for the new version. Themed and longer identifiers also work quite well.

A similar method can be used for exam randomization, where students get different copies of the exam with the same problems, but different variable names. An example from that is having CC-BY-NC-SA, PublicDomain, CreativeCommons, or YouTube-StandardLicense for a variable name that students have to write or having PaperReam, GumPack, or PencilPouch as a variable to use in a problem. Long and distinct was what we found that worked best, but we took care not to make identifiers too distracting or ruin problems by changing them.

4.3 Assignment Workflow

Now that the changes to the assignment were made, and students finished the assignment, the next step was to search student code for past semester solutions. Our tool downloaded and parsed all student code. By combining Python with Google Spreadsheets, the tool was able to skip over students who might have a valid reason for containing past semester code, as discussed in 4.1.

We then searched the student code for past semester identifiers, constants, or logic by specifying what code snippits to find in the student code submission history. Everything found was uploaded to a google sheet to allow for multiple people to easily review what was flagged.

When reviewing backups, we decided to skip over any student code that was only slightly suspicious and only pursue cases that seemed obvious and egregious to us. We also looked at the code snapshots before and after the flagged backup, as well as time between backups.

4.4 Notifying Students

After flagged backups are reviewed, it is time to notify students. To save time on email writing, as part of our tool [9] we had a way to generate emails based on what the student was flagged for. The email explained what assignment the student was flagged for, and asked the student to fill out a form, example in C explaining what happened. About 60% of students admitted and apologized at this point, while the rest denied committing any sort of academic misconduct.

For the students who denied committing academic misconduct, if it was a real false positive where the student had a valid reason for having this past semester code show up in their backups, then we let the student know that. Otherwise, then we sent over all the evidence to the student, leading to a discourse over email.

For repeat offenders, we had harsher penalties. In Fall 2020, our penalty structure was zero on assignment for first offense, zero on assignment and 2/3 letter grade deduction for second offense, and failing the course for a third offense. A next offense was when a student had a backup containing past semester code after we had notified them for the previous offense.

4.5 Improving from Semester to Semester

The second semester utilizing Active Academic Integrity was different than the first. Most solutions found online for our course are many years old, and we found students mostly copied those as opposed to the most recent semester's solutions. So instead of making as many changes as we did the first semester, we made fewer changes. Instead we reviewed what changes had a high false positive rate and updated those changes and related problems.

4.6 Drawbacks

There are some drawbacks with using Active Academic Integrity. The main one is the tool is not designed to catch students who over collaborate with another student from a current semester and share answers. They would not have any past semester code. There is a straightforward solution to this problem. Utilize Moss and TMOSS instead, which both are designed to identify instances of peer to peer over collaboration. In this sense Active Academic Integrity and Moss and TMOSS are complementary, since they strive to identify different types of academic misconduct. In our case, running Moss identified only several students who seemed to have over collaborated on the projects, and was not effective at all at identifying if students over collaborated on homework assignments, since the homework assignments were so short, that two students having identical code when solutions to problems are only a few lines long was not unlikely.

Chapter 5

Results

Hundreds of students have been caught using past solutions in CS 61A in each of the past few semesters, none of whom would have been caught if it were not for our effort building Active Academic Integrity. These students are now being notified in one of their first computer science courses. Not alerting these students may lead to bad habits, being caught in a later course, or never being caught at all.

In the figures and table below, *flagged* means a backup was manually reviewed, *emailed* means student got a notification saying they have been flagged for cheating, and *penalty enforced* means the student received a penalty for that assignment.

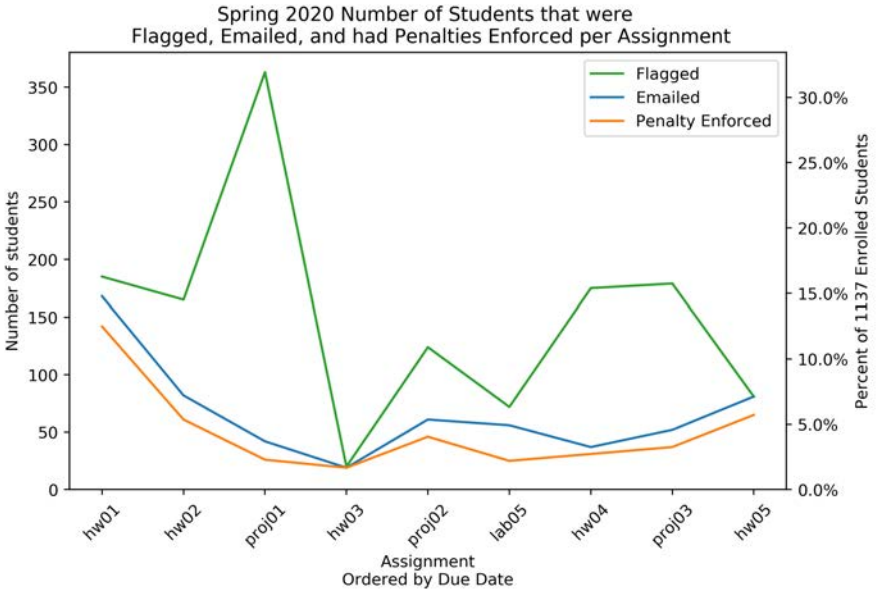

FIGURE 5.1: Spring 2020 Data Graph comparing how many students were flagged, emailed, and had penalties enforced for each assignment. Notice that Homework 1 had the largest percent of students who were emailed and had penalties enforced, showing that many students cheat as early as the first assignment.

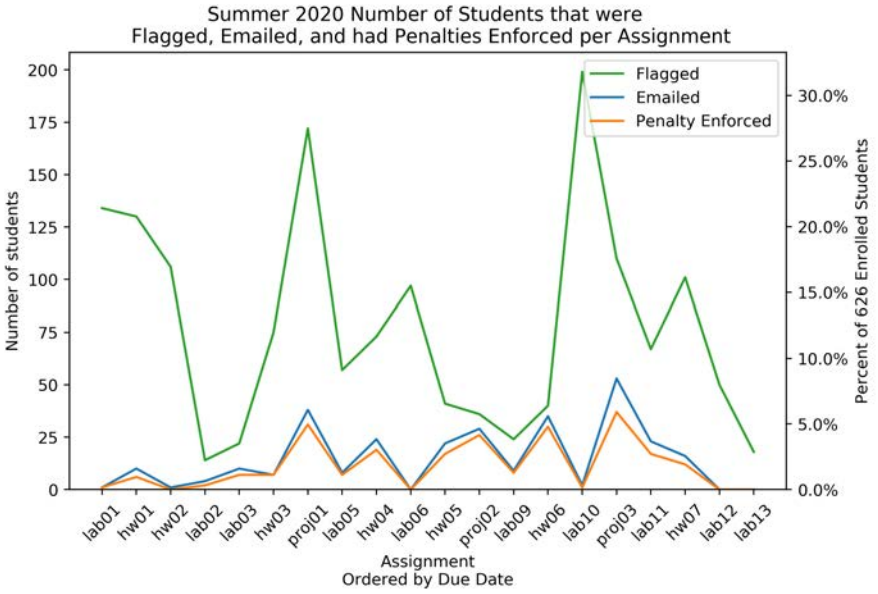

FIGURE 5.2: Summer 2020 Data Graph comparing how many students were flagged, emailed, and had penalties enforced for each assignment. This semester the instructors preferred a more conservative approach of emailing students, and many suspicious cases were not followed up with.

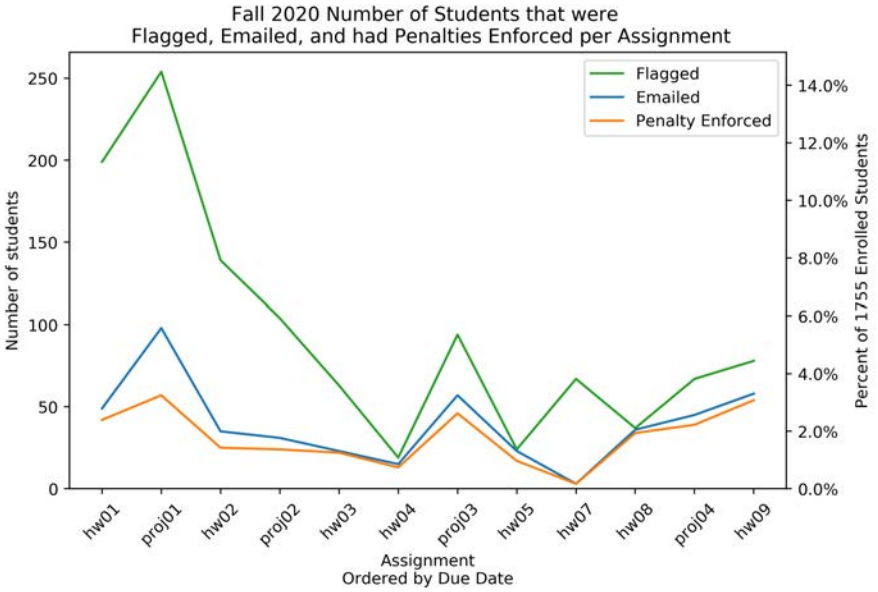

FIGURE 5.3: Fall 2020 Data Graph comparing how many students were flagged, emailed, and had penalties enforced for each assignment. The large gap between 98 students emailed and only 57 penalties enforced for proj01 was due to a partial change slipping through the review process, meaning both the past semester code and current semester code were both present on the project specification, leading to students having that code but not committing any plagiarism.

Taking the first homework as an example, we flagged and then accused 164 students out of the 1452 initially enrolled, or 11.3% of the class. Out of those who were accused, we only enforced 142 penalties. The most common reason for not penalizing students was

Semester	Assignment	Flagged	Emailed	Enforced	Email Delay in Days After Assignment Due
Spring 2020	hw01	185	168	142	2
Spring 2020	hw02	165	82	61	5
Spring 2020	proj01	363	42	26	63
Spring 2020	hw03	20	19	19	49
Spring 2020	proj02	124	61	46	42
Spring 2020	lab05	72	56	25	41
Spring 2020	hw04	175	37	31	35
Spring 2020	proj03	179	52	37	27
Spring 2020	hw05	81	81	65	23
Summer 2020	lab01	134	1	1	10
Summer 2020	hw01	130	10	6	6
Summer 2020	hw02	106	1	0	6
Summer 2020	lab02	14	4	2	5
Summer 2020	lab03	22	10	7	21
Summer 2020	hw03	75	7	7	21
Summer 2020	proj01	172	38	31	20
Summer 2020	lab05	57	8	7	18
Summer 2020	hw04	73	24	19	14
Summer 2020	lab06	97	0	0	11
Summer 2020	hw05	41	22	17	19
Summer 2020	proj02	36	29	26	17
Summer 2020	lab09	24	9	8	16
Summer 2020	hw06	40	35	30	12
Summer 2020	lab10	199	2	1	11
Summer 2020	proj03	110	53	37	10
Summer 2020	lab11	67	23	17	9
Summer 2020	hw07	101	16	12	5
Summer 2020	lab12	50	0	0	4
Summer 2020	lab13	18	0	0	2
Fall 2020	hw01	199	49	42	5
Fall 2020	proj01	254	98	57	4
Fall 2020	hw02	139	35	25	3
Fall 2020	proj02	104	31	24	10
Fall 2020	hw03	63	23	22	29
Fall 2020	hw04	19	15	13	22
Fall 2020	proj03	94	57	46	14
Fall 2020	hw05	24	23	17	11
Fall 2020	hw07	67	3	3	24
Fall 2020	hw08	37	36	34	17
Fall 2020	proj04	67	45	39	12
Fall 2020	hw09	78	58	54	3

TABLE 5.1: All data from Spring 2020 to Fall 2020 Academic Integrity Efforts

they were retaking the course and simply copied their own past solutions or they had audited the course and solved past semester assignments as a way to prepare. For any other plausible explanations we decided to not penalize students.

By the end of the Spring 2020 semester, we had sent a total of 598 notices to students, one for each lab, homework, or project a student was flagged and accused for. Out of those 598 accusations, we enforced 452 of them, or 75.6%. The human review process between what was flagged and what was emailed made sure students were only emailed if they had past semester code. There were cases, however, where students had valid reasons for having the past semester code show up in their submission history.

Example of a bad variable change:

```
def product(n, f):
    """Return the product of the first n terms in a sequence.
    n -- a positive integer
    f -- a function that takes one argument to produce the term

    >>> product(3, identity)  # 1 * 2 * 3
    6
    >>> product(5, identity)  # 1 * 2 * 3 * 4 * 5
    120
    >>> product(3, square)    # 1^2 * 2^2 * 3^2
    36
    >>> product(5, square)    # 1^2 * 2^2 * 3^2 * 4^2 * 5^2
    14400
    >>> product(3, increment) # (1+1) * (2+1) * (3+1)
    24
    >>> product(3, triple)    # 1*3 * 2*3 * 3*3
    162
    """
    "*** YOUR CODE HERE ***"
```

Here the previous second parameter to `product` was called `term` instead of `f`.

```
def summation(n, term):
    """Sum the first n terms of a sequence.

    >>> summation(5, cube)
    225
    """
    total, k = 0, 1
    while k <= n:
        total, k = total + term(k), k + 1
    return total
```

This change is bad for a few reasons. One is the word "term" is still in the problem description, and a student might then use `term` instead of `f`. The main reason is the lecture example for summation used the old variable names, so students who watched lecture and attempted to build off the lecture example to solve this problem were incorrectly emailed.

One common reason for false positives were course staff making changes that were not properly compared to lecture examples and course textbook. If students referenced the

textbook or lecture material they would copy that as an example and then be incorrectly flagged. Another reason was changes were too simple, like changing a variable name n to x that students could conceivably solve the problem with the wrong variable name and not be copying for an outside source.

At the end of Spring 2020, we had enforced penalties to 232 unique students, and the class enrollment was 1173 students.

We identified two main areas of improvement after the Spring 2020 offering. One was the false positive rate, especially after the first homework. We ideally want no false positives, since a false accusation is stressful for students. To solve this we made better variable name changes and asked the class at the beginning of the semester if they had worked on the material previously and asked them to provide their past work, as described in the Workflow and Best Practices section. The second was the sheer amount of manual work. We had some basic scripts to search backups, but to make it easier we automated uploading suspicious backup links to a spreadsheet, sending emails, and making personalized responses to those who want to argue their case.

The Summer 2020 offering was 828 students initially and then 625 students at the end of the summer. We notified students 292 times and had penalties enforced for 228 of those notifications, or 78.1%, a increase compared to Spring 2020. Software we built helped us reduce our time between the assignment due date and when we notify students from upwards of a month to being on a weekly schedule. Receiving an accusation a month after a student turned in their code is not as helpful as within a few days, and timely accusations leave more time for the students to learn from the experience and never commit academic misconduct again. Better software helped both speed up and standardize the process of flagging, reviewing, and finally notifying students.

The Fall 2020 offering started with 2054 students, ending with 1755 enrolled. We accused students 473 times and enforced 376 of those accusations, or 79.5%.

5.1 Comparing Semesters

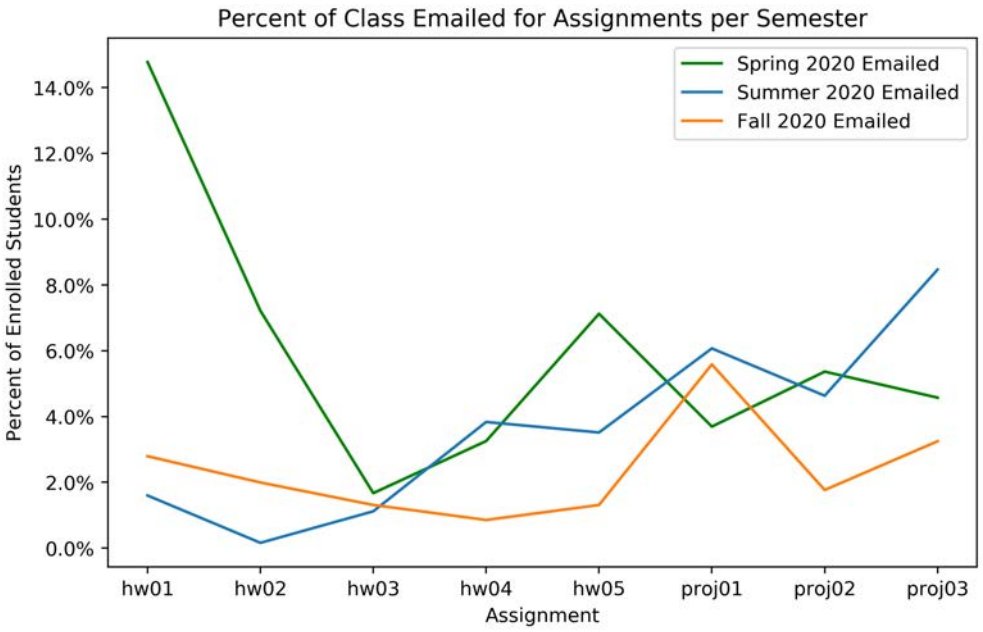

FIGURE 5.4: Comparison of Accusations

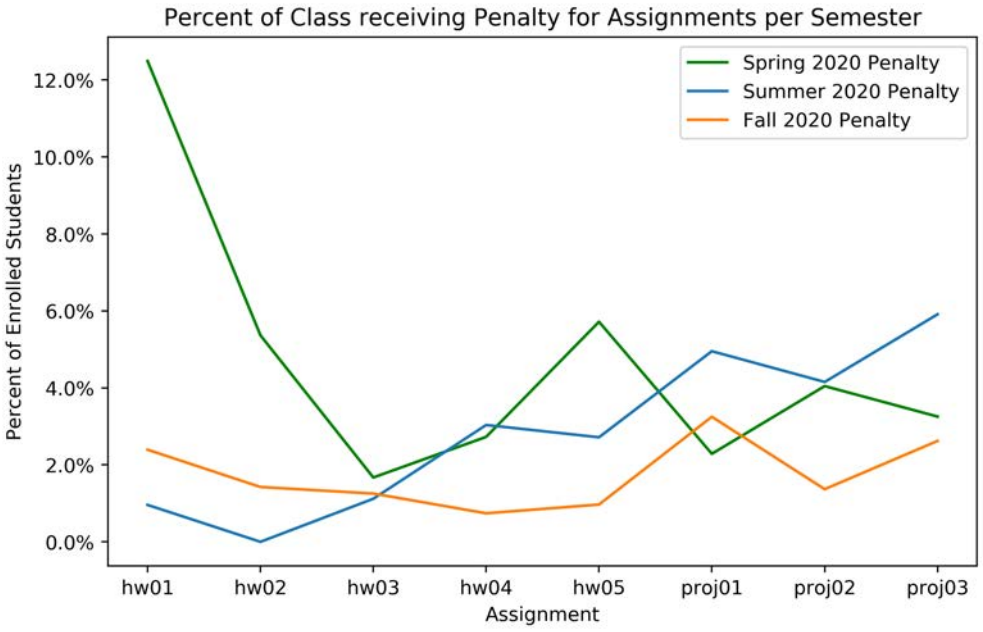

FIGURE 5.5: Comparison of Penalties

In Spring 2020 we emailed everyone who was slightly suspicious for first two assignments. Every student that had any past semester code or variable name or solution show up in their submission history was notified. Later in Spring 2020 we switched to only emailing people who had extremely suspicious backup. We only notified people who had either

full or near full solutions from a past semester show up in their backups, as opposed to partial or incorrect solutions with past semester variable names.

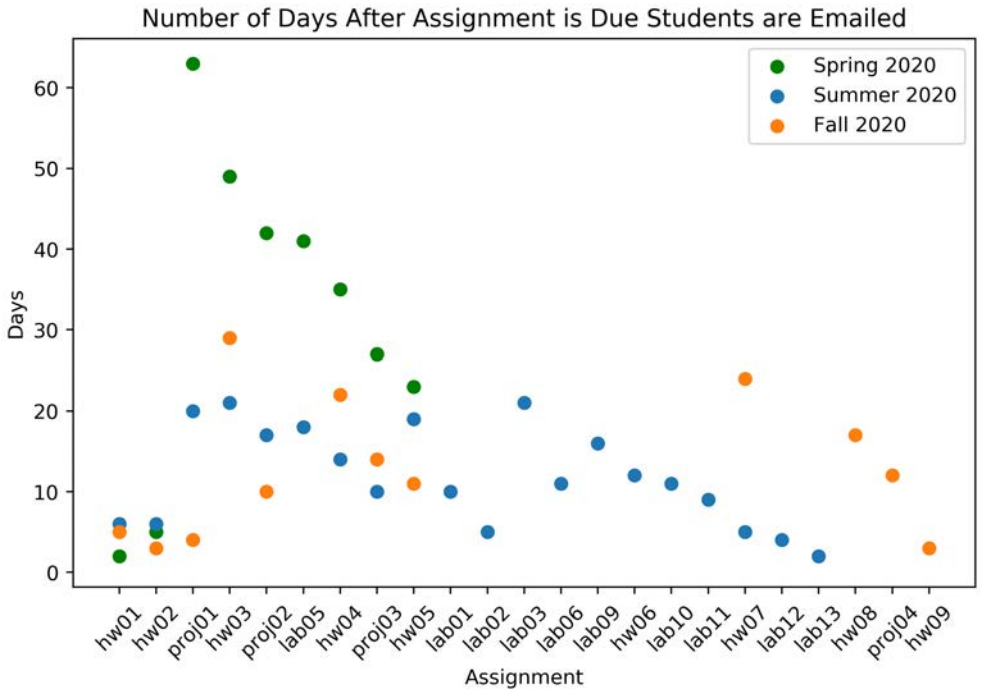

FIGURE 5.6: Comparison of Email Delay

A lot of time was spent discussing who to accuse, and reviewing backups in Spring 2020. The biggest lesson learned was that it is fine *not* to notify students about everything they were flagged for, but coming to that decision took a lot of discussion and looking at flagged student backups. Now we consider what the previous submission looked like compared to the flagged one, how likely a student is to accidentally write this wrong code if they were not using a past semester solution, whether a verbal hint from a friend might have led them to type what was flagged, and how many problems they were flagged for on a specific assignment.

In Summer 2020 we developed software to make process faster and more streamlined to review backups and send emails. Specifically the software reduced the time to parse and upload the flagged backups to a spreadsheet for one assignment from 2 hours to 10 minutes. Automating email sending allowed that portion to be reduced from 1 hour to a few seconds. The Summer 2020 software work resulted in Summer 2020 and Fall 2020 having a much faster turnaround time compared to Spring 2020, and we were able to review more assignments.

5.2 Spring 2021 Experiment: Early Email Notifications

Getting notified earlier about a potential academic integrity infraction is more likely to cause the student to change their habits. We understand that students come from very different backgrounds. The students do not all have a strong understanding of what is and is not allowed, and especially near a deadline are pressed to search for and submit work that is not their own. Whatever the reason, we want students to want to stop themselves, and an early warning realtime was our proposed solution.

Imagine a student is working hard on an assignment. It is late and the deadline is approaching. The student is struggling to debug a particularly tricky problem related to using Object Oriented Programming to create a Vending Machine. The student chooses to search "CS 61A Vending Machine solutions" online and copy a snippet from the first link, just to check their logic. While comparing that snippet to their wrong solution and testing things out, the student notices a new email in their inbox.

```
Subject: Reminder about CS61A Academic Honesty

Hi {Preferred name from welcome survey},

Our automated system noticed something suspicious as you were working on {
    assignment}. If you have any questions, please reply back to this email.

No penalties have been applied in your case. This email is just a reminder of our
     policies, as well as a set of links to resources.

The system will sometimes flag perfectly innocent code, so we don't automatically
     apply penalties. Nevertheless, we think that it might be a good idea to make
     sure you are aware of the resources we have to help you succeed in CS61A,
    and of the penalties we can apply in cases of inappropriate copying.

Piazza is the place to ask any questions you have, with regards to anything,
    especially any assignment difficulties you may have. Piazza is open at all
    hours any day, and you can even make a private post with your code that a
    staff member will get to and help you get unstuck!

Office Hours is the place to get live one on one help with a member of staff. The
     schedule of office hours is here and then you can visit oh.cs61a.org to sign
     up for the office hour queue while office hours are running.

Parties are a place to work with staff and students in a session dedicated to a
    single assignment. The schedule for parties is here.

We also want to remind you about our Academic Honesty Policy. Specifically, do
    not use solutions from other people, whether you come upon them from
    searching online or asking someone for answers. Sadly many students have been
     caught being academically dishonest and have suffered grade penalties.

Here are the penalties for Academic Dishonesty in CS61A:
```

```
    1. The first offense will be a 0 on the assignment (if homework)/questions
    flagged (if project).
    2. For a second offense, the minimum penalty is 1/3's of a letter grade
    reduction. (E.g., reducing your final grade from an A- to a B+.)
    3. A third offense can result in failing the course.

In addition, any collaboration on an exam will result in at least negative points
     on that exam.

Best,
CS61A Course Staff
```

Such a warning would likely leave a strong impression on the student, have them quickly delete all the copied code, and hopefully be forever aware of the potential ramifications of utilizing an online solution. Hopefully when tempted again to look online or ask a friend who has taken the class before for their answers, the student will remember this moment. What's more, is the student is notified on your their first offense realtime, as opposed to days, weeks, or even a month later when many more assignments have been assigned.

After receiving the automated email, students in CS 61A either responded with a "Thank you for the warning, it won't happen again", or some asked why they were emailed, and that is when we showed the specific submission backup with the past semester code, and made it clear that this semesters version of the assignment did not have that phrase or variable name or logic to solve the problem.

But it is not magic, it simply is removing the check between flagging and sending out emails to students. As the figures above showed, there are plenty of false positives in what is flagged. So in Spring 2021, we had a near real-time system that would every hour download all student backups and then automatically email students. We checked which search terms from past semesters had a high flag to notify student rate, and those we allowed an automated system to download all the backups, search for the terms, and send emails if a student was flagged for the first time. Each student got at most one automated email, and future offenses led the student to receive point penalties as opposed to just a warning.

The experiment was highly effective. Out of the 1075 students initially enrolled in the course, 246 received this automated warning email, or 22.8%.

Out of those 246 students, another 83 students again committed academic misconduct and were notified a second time after they received the automated warning. These students were given no credit on the assignments they were notified about. 5 of these cases were deemed to be false positives.

Out of those 83 students, another 11 students again committed academic misconduct and were notified a third time. These students got an additional penalty of a 2/3 letter grade reduction on top of no credit for the assignments they were notified about.

As seen in 5.6, we had large time difference between when the assignment was due and when students were notified. The automated email system greatly reduced the time between when a student potentially commits academic misconduct and a student is notified.

5.3 Anonymous Student Surveys

At the end of Spring 2020 and Fall 2020, we added an optional survey to the end of the final survey titled "Anonymous form to help improve CS 61A with regards to academic honesty" and asked students to fill it out. We had 265 responses in Spring 2020, 74 responses in Fall 2020, and 29 responses in Spring 2021. Below are some graphs aggregating student answers.

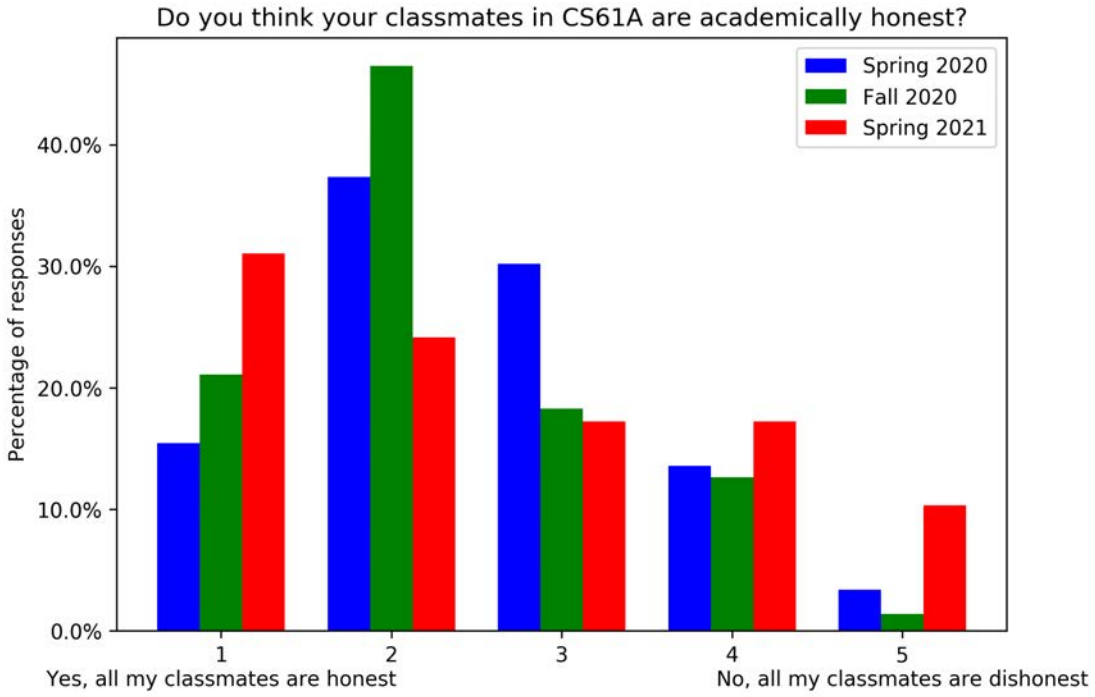

FIGURE 5.7: "Do you think your classmates in CS61A are academically honest?" Spring 2021 had the highest percentage of responses selecting both 1 for all classmates are honest and 5 for all classmates are dishonest

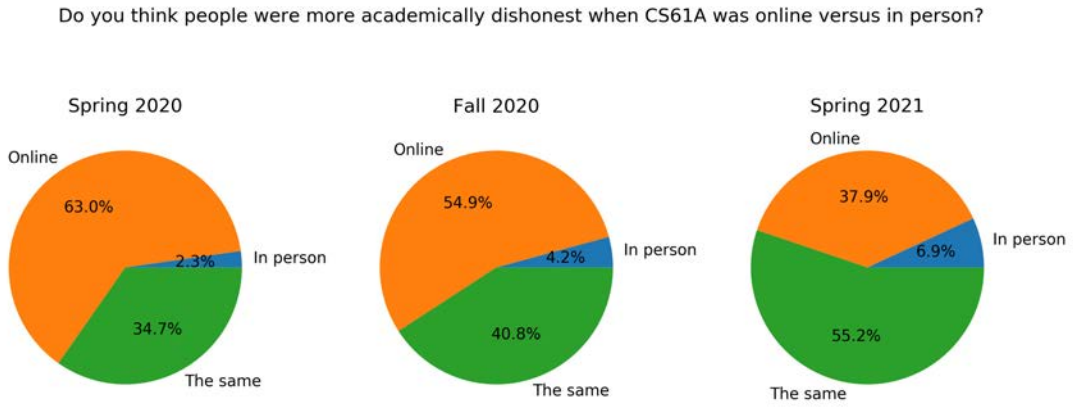

FIGURE 5.8: "Do you think people were more academically dishonest when CS61A was online versus in person?" The switch in Spring 2021 to the majority selecting "The same" might be due to Berkeley courses being online for over a year at that point in time.

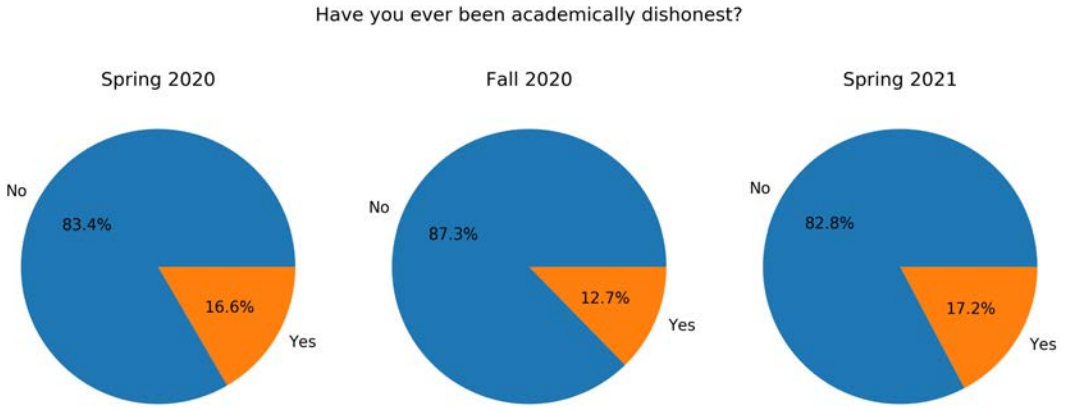

FIGURE 5.9: "Have you ever been academically dishonest?"

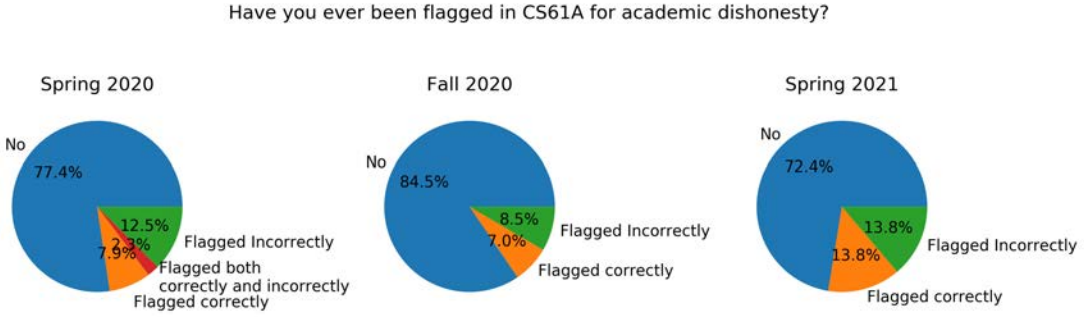

FIGURE 5.10: "Have you ever been flagged in CS61A for academic dishonesty?" The word flagged here is the same as *emailed* used in the Results section.

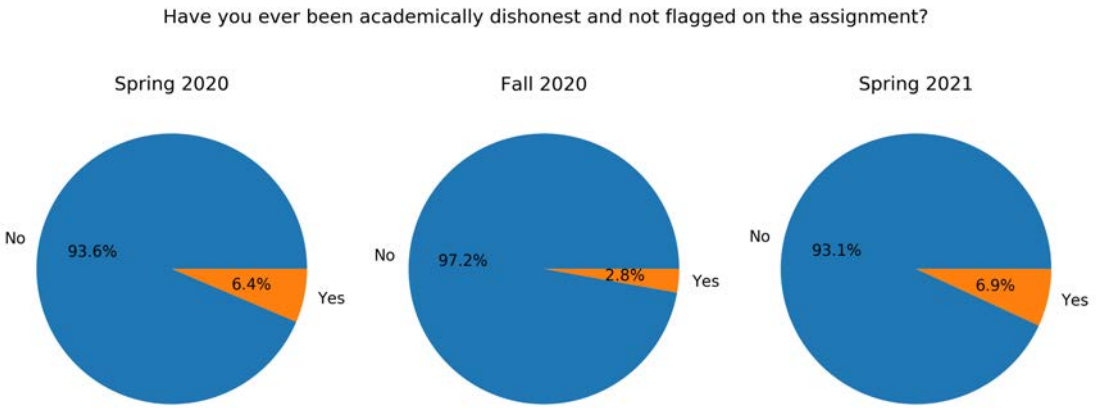

FIGURE 5.11: "Have you ever been academically dishonest and not flagged on the assignment?"

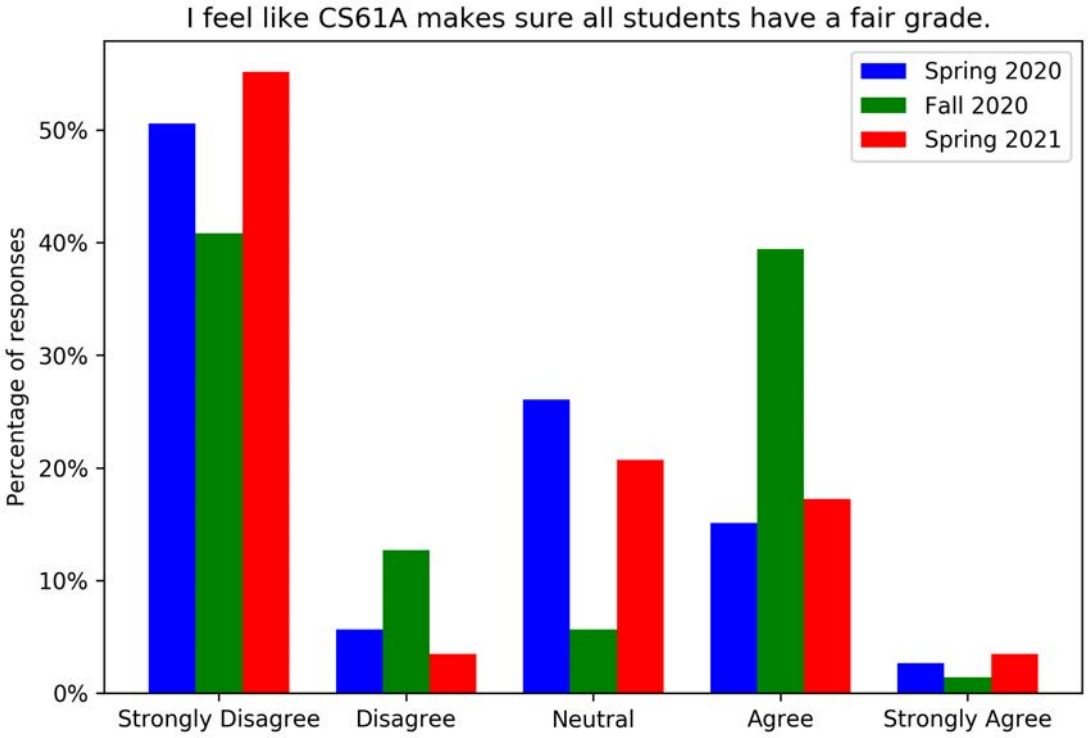

FIGURE 5.12: "I feel like CS61A makes sure all students have a fair grade." For context, students in the fall semester on average perform better than students in the spring semester. The grade a student receives in CS 61A affects their chances of studying Computer Science at Berkeley.

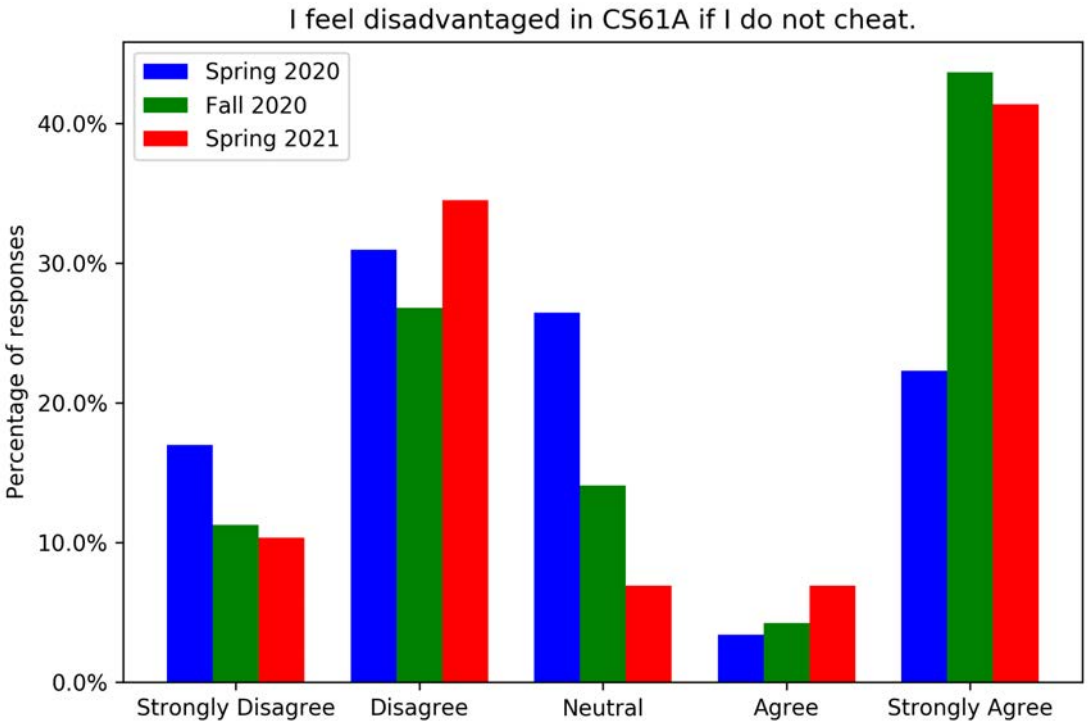

FIGURE 5.13: "I feel disadvantaged in CS61A if I do not cheat." There is a high percentage of students who feel disadvantaged by not cheating. This likely contributes to the hundreds of academic integrity cases.

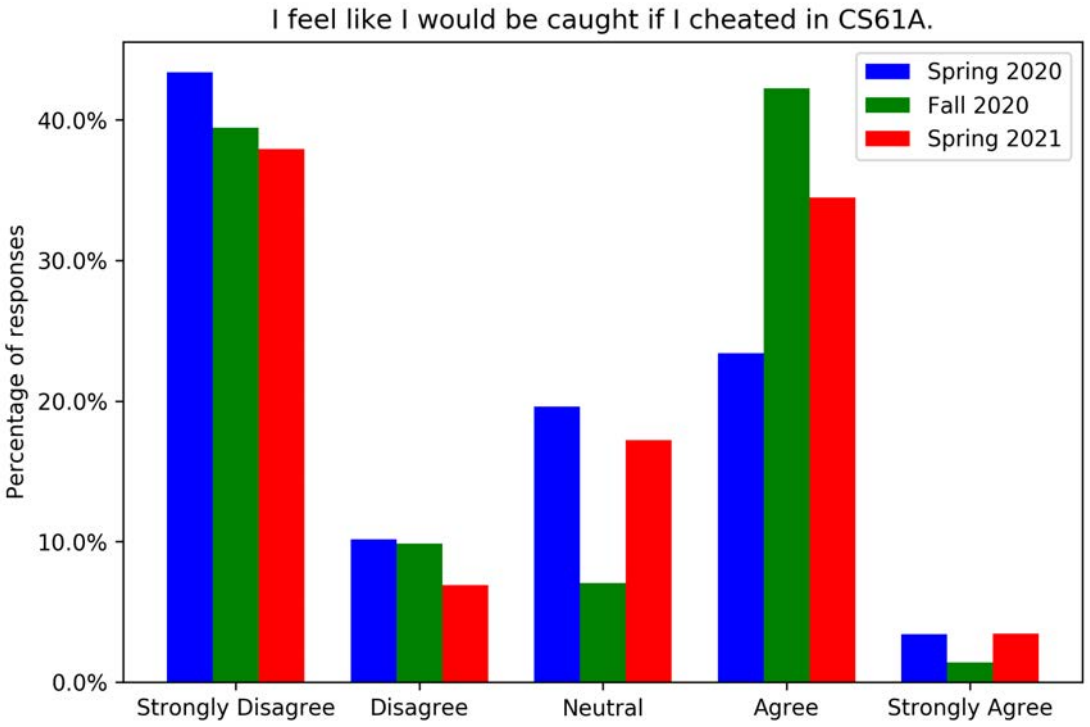

FIGURE 5.14: "I feel like I would be caught if I cheated in CS61A." Students are fairly confident in their ability not to get caught cheating.

5.3.1 Student Feedback

Here are a few select quotes from the anonymous surveys:

"I was flagged on an early homework which I did indeed violate academic honesty on, but I received an email about it about 1-2 months later. This is too long, and I think it'd be wiser to grade assignments sooner and notify students about potential academic dishonesty, because they would be much more likely to stop violating the academic honesty policy (if they were copying code from others, from websites, etc.), and this would not only help them learn more effectively by actually having to do the assignments, but also prevent them from committing further violations and potentially worrying about other assignments they may not have been honest on, but were dishonest on them before getting the notification that their work was flagged." - student from Spring 2020

"Let up on dishonesty. It shouldn't be considered dishonest to browse stack overflow. 99% of professional programmers do this on a daily basis. In fact, knowing how to Google is one of the most useful skills you can have as a developer. There should be no reason why we can't do the same in this course." - student from Spring 2020

"I cheated once and I got caught for it. I think you guys are really good at what you do. Thank you for giving me the wake up call I needed." - student from Spring 2020

"I've never been compelled to make academically dishonest decisions in THIS class..." - student from Fall 2020

Chapter 6

Conclusion

With a new method to detect cheating, hundreds of students were caught. Iterations and improvements in policy and scripting lead to reducing False Positive Rates every semester, from 24.4% in Spring 2020 to 21.9% in Summer 2020 to 20.5% in Fall 2020 to 6.02% in Spring 2021. Active Academic Integrity is complementary to Moss, and in some cases better. Notably, it is better when assignments have solutions that are only a few lines long. This system does catch students who copy past semester solutions, and improvements in software helped reduce turnaround time, which is good for encouraging students to change bad habits earlier.

6.1 Discussion Items

How remedial should an academic integrity process be, and how harsh should punishments be for getting caught cheating?

Is this system only catching students because it is novel?

Should evidence be shown to student when they are emailed?

One drawback of this system compared to Moss is that it will never catch a student copying/over collaborating with a current semester classmate. Is using an online/past semester solution worse than getting answers from a current classmate?

What should be done when a student claims to coincidentally use a past semester variable?

Chapter 7

Contributions and Future Work

All our work is open source https://github.com/alexkassil/active_academic_integrity and we hope other educators can adopt it for their courses. One planned area of extension is to have the system work seamlessly with git as well as okpy. Finally making a website interface instead of just command line scripts and spreadsheets would make the tool more accessible to others.

Appendix A

Syllabus Quiz Academic Integrity Questions

Refer to https://inst.eecs.berkeley.edu/ cs61a/fa20/articles/about.html#academic-honesty to answer questions from this section

Alyssa and Ben are students in 61A unless otherwise noted. Select all scenarios that VIOLATE academic honesty

- ☑ Alyssa and Ben work on a HOMEWORK assignment together. They work on it together on the same screen or send each other their code and as a result, have similar answers
- ☐ Alyssa and Ben work on a HOMEWORK assignment together. They share ideas and talk through it together but they do not share answers or look at each other's screens
- ☐ Alyssa and Ben work on a PROJECT together, and they are project partners for this project. They talk through it and work on it together on the same screen, and as a result have identical answers
- ☐ Alyssa and Ben work on a HOMEWORK assignment together. Alyssa finishes first and helps Ben finish by looking at Ben's code
- ☑ Alyssa and Ben work on a HOMEWORK assignment together. Alyssa finishes first and helps Ben by showing Alyssa's code to Ben
- ☑ Alyssa is a 61A student this semester and Charles was a student in Summer 2020. Alyssa asks Charles for help on a question, and Charles helps Alyssa by looking

at his solution and walking her through his solution verbally, but doesn't send his code to Alyssa

☑ Ben searches online for 61A homework solutions, only to compare to his code and figure out why his own code is wrong, not copy completely.

Any Previous CS61A Work?

Some students either are taking CS61A for another time after dropping or have done some of the assignments in the past. It is totally fine to reference your past work, but please let us know how far you got into the assignments so you do not get flagged for academic misconduct by our cheating detection software.

If you do not have access to your previous solutions or you do not plan to use your previous solutions, please answer no for this question.

Have you worked on any CS61A assignments in the past?

- Yes
- No

Previous CS61A Work

Having already done some of the material in CS61A in the past does make redoing that material easier. While you are allowed to look at your own past answers, we recommend you try each problem from scratch without referencing your past solutions, since students who retake the course or have already done the assignments learn more by trying the assignments from scratch. Also assignments change every semester, so directly copy and pasting your previous semester solutions sometimes does not work at all!

Upload your past work here

Please put all your .py/.sql/.scm files in one folder, zip that folder, and upload it here, do not upload extras like ok/scheme editor/tests/guis.

Appendix B

Referencing Previous Work/Working Ahead on Assignments Form

Some students, either due to repeating the course or previewing the material beforehand, or getting a leg up on an assignment before it is released have their own answers/work from a previous semester.

What is not ok is using a friends past semester work and claiming it is your own.

Please select which assignment you are using previous work from and upload the file from the assignment.

For most assignments just upload the one .py/.scm/.sql file

For the scheme project just upload scheme.py and questions.scm by filling out the form twice.

This form is only for those who have more assignments they worked on after submitting the syllabus quiz. For example to get ahead you worked on last semester's ants project before it was released

Which assignment did you work on?

- List of Assignments

Upload the .py/.sql/.scm file that is your own work that you will be referencing

What semester is the assignment from?

- List of Semesters

Space for adding any extra information you might want to share with the staff

Appendix C

Use of Solutions Form

Which assignments were you flagged for?

- List of Assignments

Do you agree that accessing solutions to CS 61A assignments, whether they are currently or formerly posted to the web or asking someone who has previously or currently taken the course for answers is not allowed and a violation of the academic honesty policy of the course? This is not an admission of guilt, but just a confirmation of the course policies.

- Yes
- No

Do you admit that you have violated the academic honesty policy of the course in some way? We want you to tell the truth. That being said, if you believe you're not guilty, don't hesitate to mark "no" and explain yourself in the box below.

- Yes
- No

Is there anything else you'd like to say/share?

Bibliography

[1] Alex Aiken. A system for detecting software similarity. https://theory.stanford.edu/~aiken/moss/. Accessed: 2021-05-02.

[2] Lisa Yan, Nick McKeown, Mehran Sahami, and Chris Piech. TMOSS: Using Intermediate Assignment Work to Understand Excessive Collaboration in Large Classes. *SIGCSE '18: Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 110–115, February 2018. doi: 10.1145/3159450.3159490. URL https://stanford.edu/~cpiech/bio/papers/tmoss.pdf.

[3] S. Schleimer, D. S. Wilkerson, and A. Aiken. Winnowing: Local Algorithms for Document Fingerprinting. *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 76–85, 2003. doi: 10.1145/872757.872770. URL http://theory.stanford.edu/~aiken/publications/papers/sigmod03.pdf.

[4] Jr. Bernard E. Whitley. Factors Associated with Cheating among College Students: A Review. *Research in higher Education, Vol, 39, No. 3*, pages 235–274, 1998. doi: 10.1023/A:1018724900565. URL https://link.springer.com/content/pdf/10.1023%2FA%3A1018724900565.

[5] Mark G Simkin and Alexander McLeod. Why do college students cheat? *Journal of Business Ethics*, 94(3):441–453, 2010. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.620.5105&rep=rep1&type=pdf.

[6] Dana Sheahen and David Joyner. Taps: A moss extension for detecting software plagiarism at scale. In *Proceedings of the Third (2016) ACM Conference on Learning @ Scale*, L@S '16, page 285–288, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450337267. doi: 10.1145/2876034.2893435. URL https://doi.org/10.1145/2876034.2893435.

[7] Ok: Automate grading & personalize feedback. https://okpy.org/. Accessed: 2021-05-02.

[8] Cs 61a: Structure and interpretation of computer programs. https://cs61a.org/. Accessed: 2021-05-02.

[9] Active academic integrity source code. https://github.com/alexkassil/active_academic_integrity. Accessed: 2021-05-02.