TSA SOFTWARE DEVELOPMENT

TSA State Conference 2020

SeaTac, Washington

Team ID Number:
#2093-1

Table of Contents

I. Research about the Problem……2

II. Description of our Project……4

- a. The Problem……4
- b. Solution for the Problem……4
- c. Value of Solution……5

III. Plan of Work Log……7

IV. Documentation of Software Development Process……9

- a. Project Requirements……9
- b. High-Level Software Design……10
- c. Testing of Solution……11
- d. End-user Product Documentation……15

V. Self-Evaluation/Future Prospects……19

VI. References……21

Research about the Problem

As the high school and college schooling experience continues to become more cutthroat, academic integrity has shown to be fading away. In the past, cheating was a method used to get by, a way to simply pass. Today, academic dishonesty has become the go-to for above-average college-bound students as well. In a study conducted by Rutgers University involving over 17,000 Graduate students, 17% admitted to cheating on tests, and 40% confessed to cheating on written assignments. Conversely, of 71,300 Undergraduate students surveyed, 39% noted that they cheated on tests, and a whopping 62% admitted to cheating on written assignments. However, this issue extends beyond just the higher levels of education. In a recent survey by the Josephson Institute Center for Youth Ethics, of 43,000 high school students selected from both public and private schools, 59% of students admitted to cheating on a test during the last year. Of this 59%, over half (58%) admitted to doing it more than two times. Further, one out of three high school students admitted that they used the Internet in some fashion to plagiarize an assignment. Sadly enough, it doesn't stop here. According to a survey of middle schoolers, 2/3 of respondents reported cheating on exams while 9/10s admitted to copying another's homework. Of all the courses offered, courses falling under math and science tend to be the subjects where the most cheating occurs. With online calculators, there is no incentive for students to do their work. The problems of cheating extend past the four walls of school. Some research even suggests that academic cheating may be associated with dishonesty later in life. In a 2007 survey of 154 college students, Southern Illinois University researchers found that students who plagiarized in college reported that they viewed themselves as more likely to break rules in the workplace, cheat on spouses and engage in illegal activities. A 2009 survey, also by the Josephson Institute of Ethics, reported that people who cheat on exams in high school are three times more likely to lie to a customer or inflate an insurance claim compared with those who never cheated. High school cheaters are also twice as likely to lie to or deceive their boss and 1.5 times more likely to lie to a significant other or cheat on their taxes. The problems of cheating and academic dishonesty aren’t confined to everyday school activities. This is a major issue that direly requires a solution.

Description of Our Project

The Problem

Cheating no longer carries the stigma that it used to. Less social disapproval coupled with increased competition for admission into universities and graduate schools has made more students willing to do whatever it takes to secure the A. Grades, rather than education, have become the major focus of many students. Students cheat because of a pressure to succeed, which isn't going away any time soon. Additionally, with more and more students enrolling in classes that appeal to admission officers rather than signing up for classes that involve their interests, students tend to become disinterested in the material required for their classes and are more likely to cheat. The opportunity to cheat devalues the importance of learning and places an unintended importance on the letter grade or a test score, which defeats the purpose of school. However, it is incredibly difficult for teachers to enforce anti-cheating methods. Creating multiple forms for tests and homework or enacting policies against cheating are both wasteful and ineffective. No teacher can control what goes on outside the walls of their classrooms. Answers are shared, papers are exchanged, and there is no proper solution in place to prevent that.

Our Solution

To solve these problems, we created **Neumann,** named after the inventor and early pioneer of pseudo-random numbers. The user interface is simple – with a few clicks, you can create a new test, add questions, and edit them. There's no limit to how many questions you can have!

You can also choose how many questions you have on each worksheet; for example, if you have a test bank of 500 questions, you can choose to only have 20 (random) questions for each test, which allows for more flexibility. Once you save the file, you can access it as a pdf from your file system, which is well formatted and ready for students to take.

The highlight of Neumann is how customizable it is. You can choose how many copies, questions per sheet, and add questions with a few clicks of a button. The tests are always fully randomized, so no two people have the same exact order, or even the same exact questions.

Neumann was built on a python structure consisting of four main classes: the main class (which handles UI), and three helper classes which it calls. These classes include task manager, pdf generator, and file manager, which work together to create a cohesive application, which is displayed in a simple format to the user through a markup file.

Overall, Neumann provides a simple and streamlined interface for teachers to create fully customizable tests and worksheets, and eliminates the possibility of plagiarism while revolutionizing education.

Value of the Solution

Our solution is of value because it boosts academic learning ability overall, while also making the job of teachers easier. As brought up earlier, one of the largest problems for teachers is the need to make individualized worksheets, as this takes a long time, time that could easily be spent elsewhere. By creating a program that requires the teachers to only enter their problems once, they do not have to spend time creating individual, random worksheets. The reduction of this menial task from making 30 worksheets to one problem bank will save teachers dozens of hours, allowing them time to concentrate elsewhere, such as on developing their curriculum, which is much more important. The solution also benefits students by reducing the possibility of cheating. Many times, teachers use the same worksheet rather than go through the aforementioned process of making random individual ones. By creating a program that creates randomized worksheets for students, they are not able to cheat, which would only negatively impact their own learning. Overall, by allowing teachers to save a lot of time which they can use to boost learning elsewhere, and reducing the potential of students cheating, the solution advances academic learning ability.

Plan of Work Log

Date	Task	Time Involved	Team Member(s) Responsible	Comments
11/15/2019	Each team member must research and come up with five ideas. Discuss and narrow down to one.	5 hours	All team members	Start looking for any idea regarding education
12/15/2019	Start researching individualized learning	4 hours	All team members	Research individualized learning
1/15/2020	Research common areas where individualized learning is failing	4 hours	All team members	Teachers do not have enough time
1/15/2020	Begin portfolio	2 hours	SK	Start describing problem
1/22/2020	Research background information and statistics	2 hours	All team members	Any pertinent information regarding individualized learning
1/22/2020	Begin work on software	2 hours	DS	Start creating Python program
1/29/2020	Continue working on portfolio	1.5 hours	SK	Describe solution in depth
2/5/2020	Continue working on portfolio	1.5 hours	BB	Describe alternative solutions and why they were rejected
2/12/2020	Continue working on portfolio	2 hours	AP	Value of our solution
2/19/2020	Continue working on portfolio	1 hours	SS	Start putting together and formatting works cited
2/26/2020	Test first iteration of program	2 hours	DS	Identify bugs and debug
2/27/2020	Keep working on program	5 hours	DS	None
3/4/2020	Test program with real people/interviews	3 hours	AP, SK, BB	Interview people about thoughts on project and usability
3/11/2020	Finishing touches to documentation	3 hours	SK, DS	None
3/17/2020	Finish portfolio	3 hours	All team members	Proofread
3/18/2020	Complete all final adjustments to portfolio	5 hours	All team members	DS specifically did the video demo

Advisor signature [illegible]

Software Documentation

Project Requirements

Upon determining the solution to our problem, we laid out a series of requirements to guide our software development process. These goals allowed us to streamline our design pipeline and keep user experience at the heart of the project.

Cross-Platform Capability

While most school districts in our area use Windows computers, we shouldn't exclude Mac or Linux users from using our software. Including Multiplatform support at the beginning of our process allowed us to cater to all teachers without sacrificing development time.

Lightweight Design

Teachers aren't always known for having state-of-the-art machines. By limiting the size of the program to 45 MB, including dependencies, we ensure that it can run with limited RAM.

Intuitive User Interface

Teachers have varying levels of technology expertise, so our program has to be intuitive by nature. Regularly testing the program for usability strengthens the project. Testing usability with real teachers serves a metric of success during the final stages of the software process.

File Saving Capability

The ability to save their work dramatically enhances the user's experience. By being able to save assignments and import them later, this opens the door for teachers to be able to save time by reusing questions and collaborating with others by sharing files.

Easy Export Functionality

Users should be able to use a single set of questions to generate any number of individual worksheets, each one being different and formatted as a PDF for easy printing.

High-Level Software Design

Language – Python 3.7

We chose Python 3.7 as the language of our project due to its cross-platform capability and support for a wide array of libraries. It allowed us to hit every single project requirement.

Modular Design

We separated code into four different components, each in its own file, reducing boilerplate coding. This maintained readability and allowed us to test each part independently.

Main Class

This component contained the user interface and handled the many screens that the users would see. The use of "widgets" allowed for dynamic menus. In the menu, the user can both create new worksheets and import existing ones.

Worksheet Manager Class

This component handles the content associated with worksheet made in "Main" including checking its validity. Any modifications to the worksheet are done here.

File Manager Class

This component is called by "Main" to save newly creating worksheets and import pre-existing worksheets. During saving, the worksheet is serialized into a JSON format which is later deserialized when imported for editing in the user interface.

PDF Generator Class

This component is called by "Worksheet Manager" to export a given worksheet into multiple randomized worksheet PDFs, each with differently arranged questions.

Graphics Library – Kivy 1.11.0

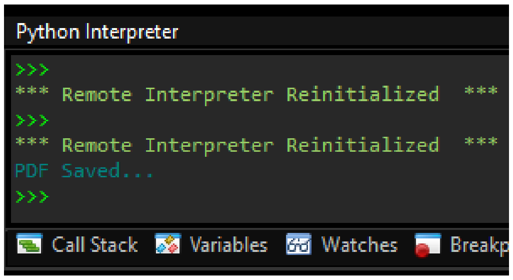
The package used to code the UI due to its cross-platform support and lightweight design.

Testing

Testing the Code

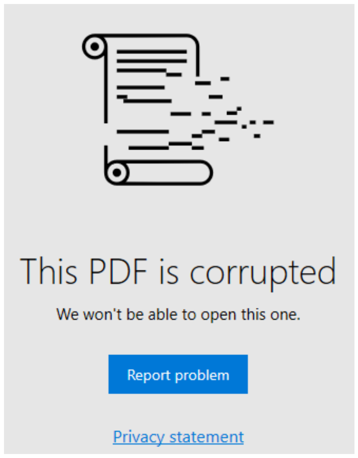
Due to our modular design, it became easy to test each component independently before putting them together. We tested on three levels: the function level, the class level, and the program level.

PDF Generator



```
>>>
*** Remote Interpreter Reinitialized  ***
>>>
*** Remote Interpreter Reinitialized  ***
PDF Saved...
>>>
```

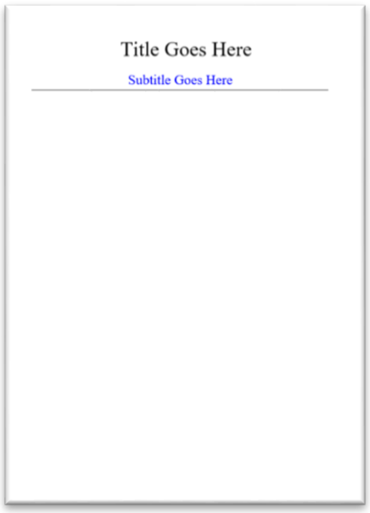
The PDF Generator was the first component we made. Our desired results were to produce a function that could be used by itself without a dependency on any of the other components. The goal was being able to generate a PDF worksheet when given a title, subtitle, and list of questions. We began by experimenting with building PDFs in Python. These first iterations reached the end of the program without errors (as shown on the right)



This PDF is corrupted

We won't be able to open this one.

. However, the PDFs generated were corrupted (as shown on the left). It turned out that the reason those PDFs were corrupted was because they didn't actually contain any elements. We made sure that these elements were added to the PDF. We moved elements around slightly and modified font size/coordinate locations to suit our aesthetic needs. The produced PDF is shown on the right. So far we had title and subtitle down, which was 2/3 of the elements we needed to have in our



Title Goes Here

Subtitle Goes Here

History Assignment #52

By Mr. Doe

How much wood could a woodchuck chuck if a woodchuck could chuck wood?

What is the eighth planet from the sun?

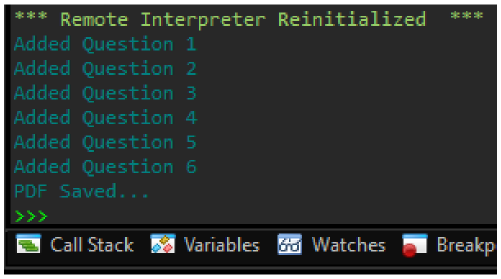
What is the eighth planet from the sun?

What is the eighth planet from the sun?

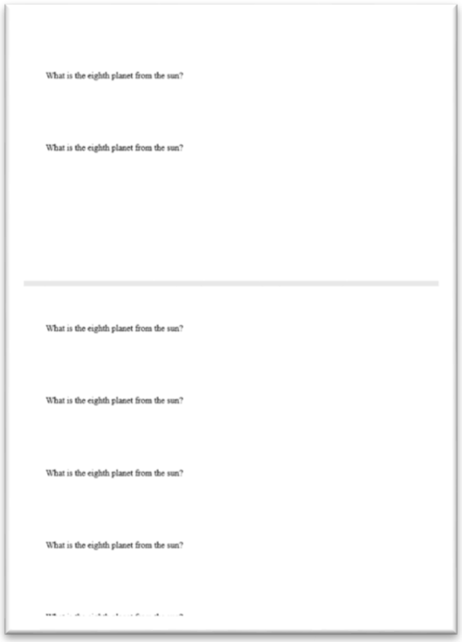
What is the eighth planet from the sun?

What is the eighth planet from the sun?

worksheet. We added a loop which added questions to the worksheet when given a list of questions, producing the following code output (shown on the right) and PDF (shown on the left). We also tested edge-cases such as when multiple pages are needed (shown on the right).



```
*** Remote Interpreter Reinitialized  ***
Added Question 1
Added Question 2
Added Question 3
Added Question 4
Added Question 5
Added Question 6
PDF Saved...
>>>
```



What is the eighth planet from the sun?

What is the eighth planet from the sun?

What is the eighth planet from the sun?

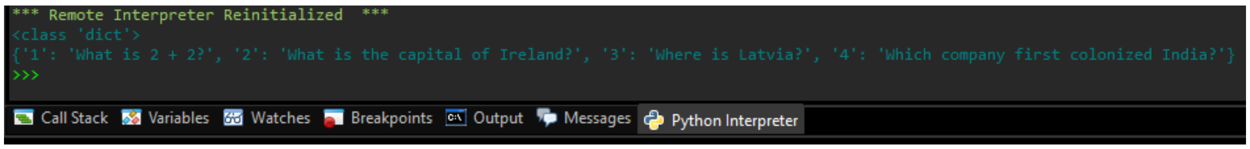
What is the eighth planet from the sun?

What is the eighth planet from the sun?

What is the eighth planet from the sun?

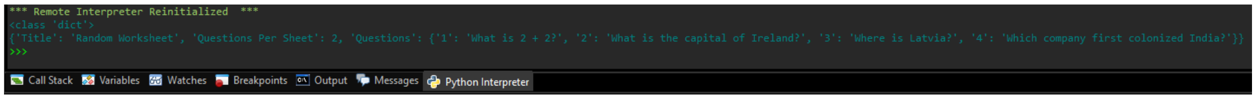
Worksheet Manager

The Worksheet Manager was the second component we designed. Our desired results were the following: able to be used by itself with only one dependency (PDF Generator), able to keep track of worksheet content (title, questions per sheet, questions), able to add more questions to the worksheet as it goes, and use PDF Generator to export the worksheets as individualized, randomized PDFs. We were successfully able to add questions to the worksheet (code output is shown below).



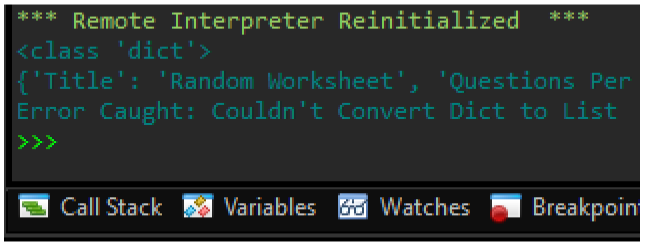
```
*** Remote Interpreter Reinitialized  ***
<class 'dict'>
{'1': 'What is 2 + 2?', '2': 'What is the capital of Ireland?', '3': 'Where is Latvia?', '4': 'Which company first colonized India?'}
>>>
```

We were also able to generate a serializable dictionary containing all three necessary pieces of information (code output is shown below).

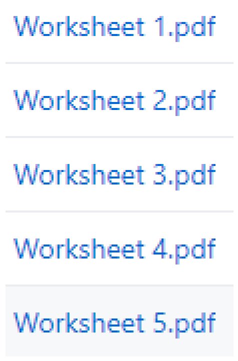


```
*** Remote Interpreter Reinitialized  ***
<class 'dict'>
{'Title': 'Random Worksheet', 'Questions Per Sheet': 2, 'Questions': {'1': 'What is 2 + 2?', '2': 'What is the capital of Ireland?', '3': 'Where is Latvia?', '4': 'Which company first colonized India?'}}
>>>
```

We then tested the PDF generating ability of Worksheet Manager. The first couple of times, it didn't run successfully and had trouble converting a larger questions dictionary to a smaller list of randomized questions (code output is shown to the right). After we fixed the errors, we were able successfully generate multiple unique PDF (shown to the left).



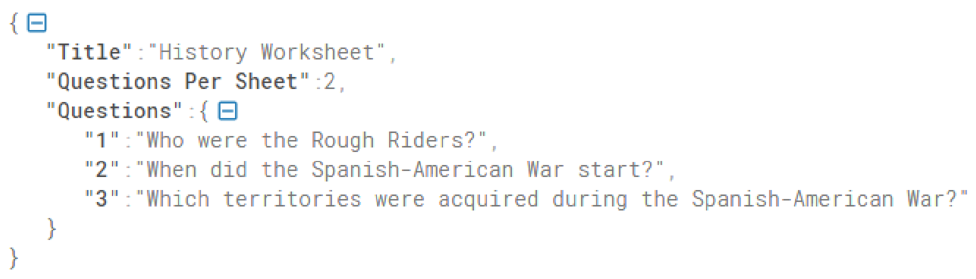
```
*** Remote Interpreter Reinitialized  ***
<class 'dict'>
{'Title': 'Random Worksheet', 'Questions Per
Error Caught: Couldn't Convert Dict to List
>>>
```



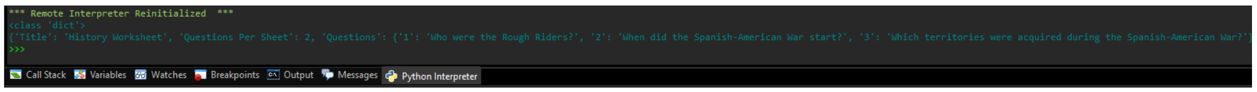
Worksheet 1.pdf
Worksheet 2.pdf
Worksheet 3.pdf
Worksheet 4.pdf
Worksheet 5.pdf

File Manager

The File Manager was the third component we designed. Our desired results were the following: able to serialize a given dictionary as JSON and able to deserialize a previously serialized worksheet to a dictionary that can be used by Worksheet Manager. We were eventually able to successfully produce a JSON output (shown on the above). We were also able to successfully deserialize the saved worksheet (code output shown above).



```
{
    "Title":"History Worksheet",
    "Questions Per Sheet":2,
    "Questions":{
        "1":"Who were the Rough Riders?",
        "2":"When did the Spanish-American War start?",
        "3":"Which territories were acquired during the Spanish-American War?"
    }
}
```



```
*** Remote Interpreter Reinitialized  ***
<class 'dict'>
{'Title': 'History Worksheet', 'Questions Per Sheet': 2, 'Questions': {'1': 'Who were the Rough Riders?', '2': 'When did the Spanish-American War start?', '3': 'Which territories were acquired during the Spanish-American War?'}
>>>
```

Main Class (User Interface)

At this point, we transitioned from the class stage of testing to the program stage of testing. We were no longer testing the functionality of a specific class and instead looked at the program as a whole. Our desired results were the same as the requirements for the project. We built a user interface screen by screen, starting with Main Menu, then moving on to Create Worksheet, Save Worksheet, and Import Worksheet. We fixed bugs as we went along.

Testing in the Field

We tested our program with real people, primarily to assess usability. We interviewed each person to learn more about their background and whether our software would be useful to them. The results are compiled in the table below.

Name (Initials)	Profession/Role	Observations During Use	Personal Comments
HS	Elementary School Teacher (3rd Grade)	• Was able to figure how to add questions without difficulty • Figured out how to edit questions without additional explanation • Had some trouble figuring out how to set the Title • Understood the file saving system • Was able to export worksheet without difficulty • Able to import a pre-existing worksheet and edit it	• Appreciated the clean/minimalist user interface • Said it was pleasant to use and "very fast" • Said they probably wouldn't use this program in the classroom since cheating isn't a problem in the 3rd Grade • Said this would probably be more helpful for middle and high school teachers
AC	High School Teacher (History)	• Spent a moment understanding the "Create Worksheet" screen before using any buttons	• Said the user interface was "powerful once you get used to it" • Appreciated the saving/importing

		• After first using the "Add Question" button, the user doesn't hesitate when making the worksheet • Continues to add questions until six total questions are made • Tries saving the worksheet before setting properties but can't • Sets properties with ease, doesn't need any guidance on figuring out how to do that • Saves worksheet in a custom location instead of the default location • Able to import that same worksheet later with ease • Exported the worksheet with ease, but was confused about where the worksheets saved	teacher and said it would be useful for collaborating with others • Said this program would be useful for them because a lot of the students in their class copy homework right before class starts • Would appreciate support for more question types, such as multiple choice

End-User Product Documentation

Run From Source (Optional)

This option allows people to recompile the program after making their own changes to the Python code. This section will get you a copy of the project up and running on your local machine for development and testing purposes.

Prerequisites

What things you need to install the software and how to install them

1. You need Python 3.7 or higher. This is the programming language the project uses. This is available on the Python website at https://www.python.org/downloads/ if you don't already have it downloaded.

2. You need Kivy 1.11.0 or higher. Kivy is the Python graphics library used in this project for the user interface. This is available on the Kivy website at https://kivy.org/#download. Follow the installation intructions for your operating system.

3. You need the latest version of ReportLab. This allows us to generate our PDF. This can be installed using pip with the following command: *pip install reportlab*

4. You need a Python IDE of your choice. Make sure it is configured to the correct interpreter. Some of the IDEs used in this project were Thonny, Python IDLE and PyCharm.

Installing

1. Clone this our repository onto your computer.

2. Navigate to the cloned repository.

3. Open the main.py file inside the Programs folder with your Python IDE.

4. Press the run button, and the program should run.

5. Edit any of the files in that folder (except .ttf files) to change the code.

Run From Executable

This is the simplest way to run the program. This method allows you to run Neumann “straight out-of-the-box.”

Opening The Program

1. Download the program folder to your computer.

2. Open the shortcut labeled “Neumann”. This will open the program.

Creating A Worksheet

1. Click on the “Create New Worksheet” button. This will take you to the Worksheet Creation Screen.
2. Click on the “Add Question” button to add a question to the worksheet. You can add as many questions as you want.
    a. If you want to delete a question, you can click on the “Delete Question” button, and then click on the question you want to delete.
    b. If you want to reset the worksheet, click “Reset Worksheet”.
3. To edit a question, click on its text button, type the text for the question, then click on the “Confirm Button.”
4. Click on the “Set Properties Button” to set the properties of the question. This should open a popup.
    a. Type the title for the worksheet. This title will be displayed on the PDF generated from the worksheet.
    b. Selected a number for “Questions Per Sheet.” This represents the number of questions that will appear on any single worksheet when a PDF is generated. This number cannot be more than the number of questions you wrote because these questions are randomly selected from that set.
    c. When you are done, click the “Confirm” button.
5. When you are ready to save your worksheet, click the “Save Worksheet” button.
    a. Type in a name to save your file as.

b. Choose the location to save your file. The “Worksheets” folder is selected by default.

Self-Evaluation/Future Prospects:

Member 1: Our team worked fantastically, collaborating on every aspect of the project and leaving no creative stone unturned. Honestly, I really feel like there wasn't a choice we made that didn't benefit the team for the better, everything just went our way!

Member 2: Despite how hard the programming and technical challenge was, we persevered through to end building a program that not only met our expectations but exceeded them.

Member 3: At least for me, team communication could've been a little bit better, as I was often confused as to what I was supposed to be doing and always felt like I was missing some part of a bigger plan. However, I was very happy with the end product so, live and let learn, I guess.

Member 4: What can I say, our team did great. There were definitely a few moments where people either weren't playing at their best and not pulling their weight, which did create some strife in the team, but we persevered and succeeded in the end.

Member 5: I did most of the backend work on the program, but from the experiences and meetings I shared with the other members, I really felt like we worked well as a cohesive unit, completing all the tasks assigned.

As seen above, we were all very happy with the outcome of our project and the work done by each member. However, there is always room for improvement and if we were to ever phase this software out to the general public, there are 3 critical items of our program that we would love to address. The first item we wish to add is support for multiple question varieties like multiple choice. These are often very common question types used by teachers on tests and adding variety would increase the functionality of our app. The second item we'd address would be adding support for more formatting options. These would be features like adjusting font, text size, spacing, color, etc. The final change that we’d implement would be to improve the more confusing aspects of the user interface, and thus streamline the entire software for ease of use. Overall, we all were very pleased with the software and the changes above do not rest and regrets in our hearts. They are simply things that must be addressed at later dates.

References:

“8 Astonishing Stats on Academic Cheating.” *OEDB.org*, Copyright © 2006-2020 OEDb - Accredited Online, Specialty, and Campus-Based Colleges, 31 Mar. 2016, oedb.org/ilibrarian/8-astonishing-stats-on-academic-cheating/.

Danilyuk, Julia. “Academic Cheating Statistics: This Is What You Ought to Know.” *Unicheck Blog for Education Junkies*, Unicheck, 21 Feb. 2020, unicheck.com/blog/academic-cheating-statistics/.

ENGR110/210: Perspectives in Assistive Technology - Academic Cheating Fact Sheet, Stanford, 12 Feb. 2017, web.stanford.edu/class/engr110/cheating.html.

“Plagiarism: Facts & Stats.” *Plagiarismorg RSS*, Plagiarism.org, 6 Aug. 2016, www.plagiarism.org/article/plagiarism-facts-and-stats.

“Statistics.” *International Center for Academic Integrity*, 21 Nov. 2019, www.academicintegrity.org/statistics/.

STUDENT COPYRIGHT CHECKLIST

(for students to complete and advisors to verify)

1) **Does your solution to the competitive event integrate any music?** ☐ YES [✓ NO]

   If NO, go to question 2.

   If YES, is the music copyrighted? ☐ YES ○ NO

   If YES, move to question 1A. If NO, move to question 1B.

   1A) Have you asked for author permission to use the music in your solution and included that permission (letter/form) in your documentation? If YES, move to question 2. If NO, ask for permission (OR use royalty free/your own original music) and if permission is granted, include the permission in your documentation.

   1B) Is the music royalty free, or did you create the music yourself? If YES, cite the royalty free music OR your original music properly in your documentation.

CHAPTER ADVISOR: Sign below if your student has integrated any music into his/her competitive event solution.

I, ______________________ (chapter advisor), have checked my student's solution and confirm that the use of music is done so with proper permission and is cited correctly in the student's documentation.

2) Does your solution to the competitive event integrate any graphics? [✓ YES] ○ NO

   If NO, go to question 3.

   If YES, is the graphic copyrighted, registered and/or trademarked? ☐ YES [✓ NO]

   If YES, move to question 2A. If NO, move to question 2B.

   2A) Have you asked for author permission to use the graphic in your solution and included that permission (letter/form) in your documentation? If YES, move to question 3. If NO, ask for permission (OR use royalty free/your own original graphic) and if permission is granted, include the permission in your documentation.

   2B) Is the graphic royalty free, or did you create your own graphic? If YES, cite the royalty free graphic OR your own original graphic properly in your documentation.

CHAPTER ADVISOR: Sign below if your student has integrated any graphics into his/her competitive event solution.

I, [illegible] (chapter advisor), have checked my student's solution and confirm that the use of graphics is done so with proper permission and is cited correctly in the student's documentation.

3) Does your solution to the competitive event use another's thoughts or research? [✓ YES] ○ NO

   If NO, this is the end of the checklist.

   If YES, have you properly cited other's thoughts or research in your documentation? If YES, this is the end of the checklist.

   If NO, properly cite the thoughts/research of others in your documentation.

CHAPTER ADVISOR: Sign below if your student has integrated any thoughts/research of others into his/her competitive event solution.

I, [illegible] chapter advisor), have checked my student's solution and confirm that the use of the thoughts/research of others is done so with proper permission and is cited correctly in the student's documentation.