Charles Bevington
Jace Boby
Steven Coraor
Daniel Giannone
Varun Patel
Jonathan Shaub

Overview

CSE-PASS is a client-server program used for online computer sciences and distance education. It works by first having a student login and verifying their information. Once logged in, the student receives a c++ problem. They then write the code to solve the problem, and submit it once they think they have the correct solution. Submitting the code sends it to the server for testing and style checking. If correct, the student can successfully submit the code and move onto the next problem. If incorrect, the student can modify the code and test again.

For security, the program takes screenshots and webcam photos of the user periodically in order to ensure that they are not using outside resources and that the user who is logged in is actually the one writing the code. When the student submits, the webcam photos and screenshots get submitted to the server along with the users code.

Received Status

At the beginning of the semester, we were provided with a Java Netbeans project containing the three source code files that make up the client-side application, named Main.java, WebcamGrabber.java, and SshotThread.java. The purpose of these files is as follows:

- Main.java - contains the logic for the application window at both the login screen and the main application screen. It handles all program events.
- WebcamGrabber.java - contains the logic for the webcam capture thread. It is started once the user is logged in and captures the user's webcam image at random intervals.
- SshotThread.java - contains the logic for the screenshot capture thread. It is started once the user is logged in and captures an image of the user's screen at random intervals.

We were also provided with root access to the adder.cse.psu.edu server, which contained the backend of the software system. It is an Ubuntu 14.04 server with a standard LAMP server installation that provides Apache2, MySQL, and PHP. The backend that the client communicates with is made up of PHP scripts that control access to the database, retrieve problems for the user, call a sandbox environment to run student code, and handle problem submissions.

The MySQL database contains the authorized usernames and passwords that can connect to the system, as well as the problem submissions from each user.

Notable Issues with Received Software:

The software was presented to us in working condition; the main features and requirements of the software were functional and were tested before any work began. However, several issues stood out to us and more became apparent during the semester:

- The code on both the client- and server-side was poorly commented, and a lot of time was spent trying to fully understand the function of each part. There was almost no documentation of how each piece of the system was linked together. Additionally, some of the code was poorly formatted, and issues such as bad indentation made it difficult to read and understand the code.
- The user interface, especially the main application screen, was somewhat poorly designed. The layout of the four textboxes on the main screen were set to a fixed size, which would cause elements of the window to appear cut off or inconsistent when viewed on various display sizes. Additionally, the placement of the boxes was somewhat unintuitive.
- Stylecheck, a server-side application that checks the user's code for style and formatting issues, was not working properly.
- finediff.php, a server-side PHP script that compares the output of the student's code versus the output expected for a problem, was overly strict regarding whitespace comparison, and would occasionally mark correct results as wrong.
- The webcam and screenshot capturing threads would not properly stop executing when the main application window was close; this would result in the threads continuing to run and capture webcam images and screenshots even after the

user thinks they have exited. The only way to fully and reliably stop the application and the threads was to force-quit the Java runtime executable.

- An issue that became known about a month into the semester was the implementation of the sandbox on the server. The previous semester configured a Sandboxie install on the server to executable student-submitted code in; however, Sandboxie is commercial software, is not open-source, and no proper license was obtained to use it. A major part of this semester was to replace this configuration with something else that had the same function but could be freely used.

Team Organization/SDLC

The organization of our teams at first was determined by the three broad categories of our system:

- User Interface
- Backend
- Systems

However, as the project progressed, we shifted away from the broad categories and created dynamic teams to address and work on the major components of the project. These included:

- User Interface (shifted from cosmetics toward functionality)
- Relocation Team - moving from PSU to commercial server
- Sandbox Team
- Queue Team
- Database Team
- Integration/Workflow Team

This style of organization suited our needs more since it allowed small groups to tackle multiple components of the system at once and then come together to integrate them. The User Interface originally wanted to develop a more modern look and feel for the client; however, the more practical features that made the system work as a whole quickly trumped the redesign in terms of importance. So, the UI shifted its goals toward functionality and integration with the new server, queue system, and pinging for server output.

Similarly, more functionality focused teams were created such as the Relocation Team. These teams took on the task of moving the existing project to a new server to allow for more flexibility and to avoid the issue of requiring the client to log into the Penn State CSE network via VPN. Relocation came with the challenges of requires new SSL for the client code and implementing all of the original functionality on the new server.

As mentioned previously, the issue of the previously used sandboxie being commercial software required remodeling a large portion of the system. We also decided to add a queueing system for run requests from the client. Both of these teams, the sandbox and queue teams, required an additional integration/workflow team to get all parts of the dissected system working together toward a common goal. This team looked at the workflow of a user's session and made it a seamless flow, from login to logout, including getting problems, running code to match against expected output, and submitting final code.

Finally, the database team made sure that both application data such as user and problem information was being stored in the database and properly retrieved, and that final submissions of student as well as screenshots and webcam images were being properly submitted.

The organization of our teams was also partly chosen by the process model we employed for the software development life cycle of the semester project. We decided to follow a modified version of the Prototyping Model as the guiding software development process for the semester project. Below is a listing of the aspects we find important to incorporate into this custom process mode:

Charles Bevington
Jace Boby
Steven Coraor
Daniel Giannone
Varun Patel
Jonathan Shaub

Prototyping Model

- We used a modern, iterative coding process that focused on high frequency deliverables
- We received a prototype of the software and developed an additional list of requirements for the prototype design phase.

Spiral Method

- We used Dr. Shaffer's essay, *Designing the ideal program assessment system to support mastery learning in an online environment*, as our Concept of Operations document
- We submitted system specifications to Dr. Shaffer that also served as a Concept of Operations document. These specifications were re-examined throughout the process phases.

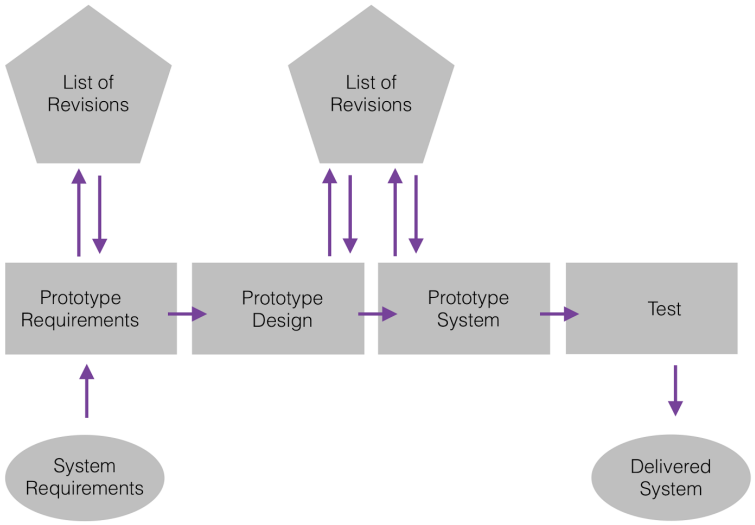
Agile Methods

- Unit testing specific to features and edge cases for fast turnaround was used throughout the prototyping design phases.

Extreme Programming

- Pair and group programming was used both in class and during out of class group meetings to collectively tackle programming problems.
- The on-site customer, Dr. Shaffer, was very helpful throughout the process.

We also developed a diagram of our customized process model which, although resembling the dreaded "waterfall" model in some ways, actually allowed for constant evaluations and revisions of our prototype systems:



The system requirements alluded to in the diagram was developed during the "requirements analysis" phase of our software development life cycle. These requirements and specifications also served as a core concept of operations document for our process model. Having that list of requirements allowed us to reassess what our project goals were for the semester during the "revision" stages of our customized prototyping model. Here are the specifications that we collectively drafted with the help of the on-site customer, Dr. Shaffer:

Security issues

- Hide and/or encrypt webcam and screenshot images
- Deal with second monitor screenshot issue
- Make sure that webcam and screenshots stop on program exit
- Handle loss of program focus (totally prevent and display error message, exit program immediately and on re-login attempt prompt to contact instructor)

Upload/Performance issues

Charles Bevington
Jace Boby
Steven Coraor
Daniel Giannone
Varun Patel
Jonathan Shaub

- Package all the program's information (including timer info) into a single file to be submitted at a later time if server connectivity is lost
- Place submitted student programs into a queue and execute from the queue instead of running all submitted programs in parallel
    - Determine a load factor, the number of parallel events, or threads, that are able to smoothly happen at once
    - Once the load factor is reached, additional submission requests are put into a randomized queue and routed to the first available thread of execution
    - Only con would be if it is not truly necessary at this point

User Interface

- Design a more user friendly and modern interface (GUI Builder Tool)
- Import student code into output box for errors with line numbers for both student input area and output area
- Clearer instructions for the student
- Improve the use of finediff.php, highlightdiff.php, and implement/modify stylecheck.cpp to satisfy our correctness checking needs.

Entry Point of Program

- Create an entry "portal" that monitors the student's computer and only continues to the runnable program if:
    - Student checks the consent box for webcam/screenshot captures
    - Webcam is working (preview shot to show correctness)
    - Internet connectivity is appropriate (download/upload speeds)
    - Background processes succeed:
        - webcam images/screenshots are being saved correctly
        - encryption is executing correctly

Admin Improvements

- Debug flag for various development issues
    - If debug is on:
        - allow copy/paste
        - disable webcam requirement (program does not quit on no webcam)
- Improve admin accounts for debugging (including Admin portal)

"Quick" fixes

- Standardize filenames for webcam and screenshot images
- Remove support for MATLAB
- Login screen enter button fix
- Cross-platform cursor issue
- Word wrap for output box
- Seconds counter
- Clear user input and output after each problem
- malloc() issue - determine server stability by attempting to run code that uses system calls, such as malloc() and fork(), to waste computational resources

Low priority Suggestions

- Add support for Java
- Add Syntax environment for student coding area
- Compiler error correction method - first error only or all at once (this can be switched easily enough that we will first deal with importing the student's code into the output box with line numbers so that the error messages are more coherent)

Charles Bevington
Jace Boby
Steven Coraor
Daniel Giannone
Varun Patel
Jonathan Shaub

By examining these system requirements, you can begin to see how the organization of our teams developed. Furthermore, the software development life cycle was naturally followed because of the process model and organization of teams that we employed. In each prototype phase, each team followed the first few stages of the software development life cycle:

- Requirements analysis
- Software design
- Implementation
- Testing

Then, all teams collectively integrated the various developed components together resulting in a deployed prototype which follows the next two stages of the SDLC:

- Integration
- Deployment

These stages were repeated multiple times during the course of the project since each prototype required this process to be followed by each of the teams. This in itself is an indication that we used and followed the spiral method of software development as part of our customized process model.

Because of the nature of the project, a semester long class, the only stage of the software development life cycle that we did not deal with was the maintenance stage. Because our role was primarily a development role, getting the software project into a usable state on a workable server, and because we will no longer be part of the project after the course is over, our project team did not need to deal with maintenance of the software project as it currently exists. However, we did keep maintenance in mind as we developed our system, both for future developers who work on the project (with improvements to code organization and commenting) as well as future users.

All of this planning and organization created a productive environment for developing a comprehensive and cohesive software project that met the system specifications outlined as goals for the semester.

Description of program

Client side

The client side of the program is everything that will need to be installed and ran by the student. The student will download a zip file containing a folder, in which will be two items. The first item is the executable file, which the student will run to start the program. The second item is a folder called SSL containing the authentication information the client needs in order to connect to the server. The executable file consists of three parts. The smaller two parts are threads that run to take screenshots and capture images from the user's webcam. The main part of the executable is the interface. When the program is first launched, the user is taken to a login page, where they must enter a valid username and password, as well as consent to have their activity monitored while using the software. Next, they are taken to the main screen, which is divided into four areas.

- Problem definition pane, which describes the problem to be solved by the student
- Example of the expected output when given a certain input
- Field to enter their code
- Output pane for any feedback on their code, which can include compiler errors, style errors, and a comparison of what the code produced versus what was expected

Charles Bevington
Jace Boby
Steven Coraor
Daniel Giannone
Varun Patel
Jonathan Shaub

These areas are all initially blank, but are filled as the information is requested. To use the program, the user must do the following steps:

1. Log in using valid credentials
2. Click "Get Next Problem" to retrieve the next problem that they must complete
3. Type the code to solve the problem
4. Click Run Problem to send it to the server and get feedback
5. Check for any style, compile, or logical errors
6. Correct any errors and go back to step 4.
7. When no errors remain, click Submit Problem to submit the assignment

Logging in obviously is a form of authentication, but also gets the user's information to provide a better experience. The user must consent to have webcam images and screenshots taken at random intervals during usage of the program; this helps to ensure academic integrity by making sure that the person solving the problems is who they say they are and that they are not looking up answers in another browser window.

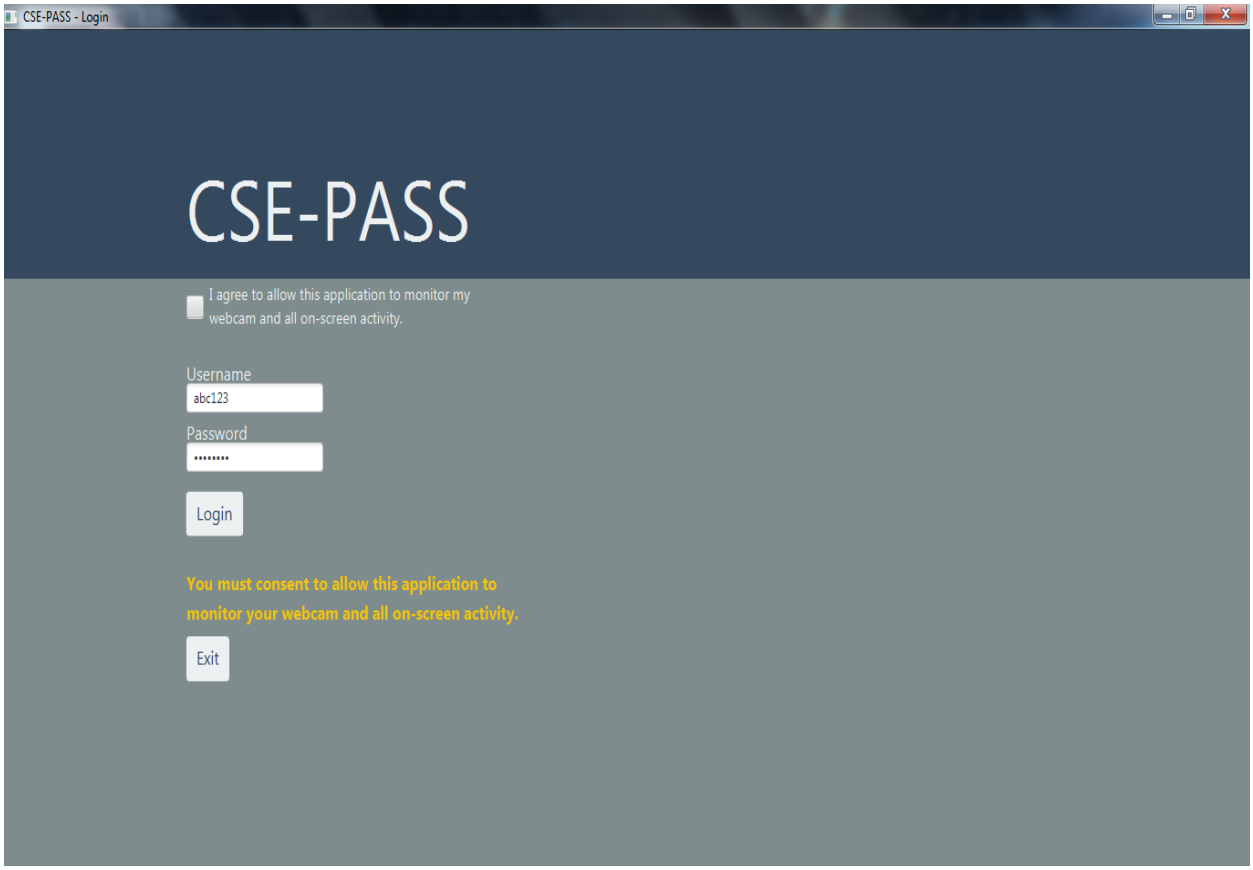


Figure 1: the login screen

Charles Bevington
Jace Boby
Steven Coraor
Daniel Giannone
Varun Patel
Jonathan Shaub

Get Next Problem retrieves the next problem to solve from the server. The next problem is determined by how many problems the student has already completed. If there are no more problems for the student to solve at that time, the program will return a message saying that no problems need to be done.

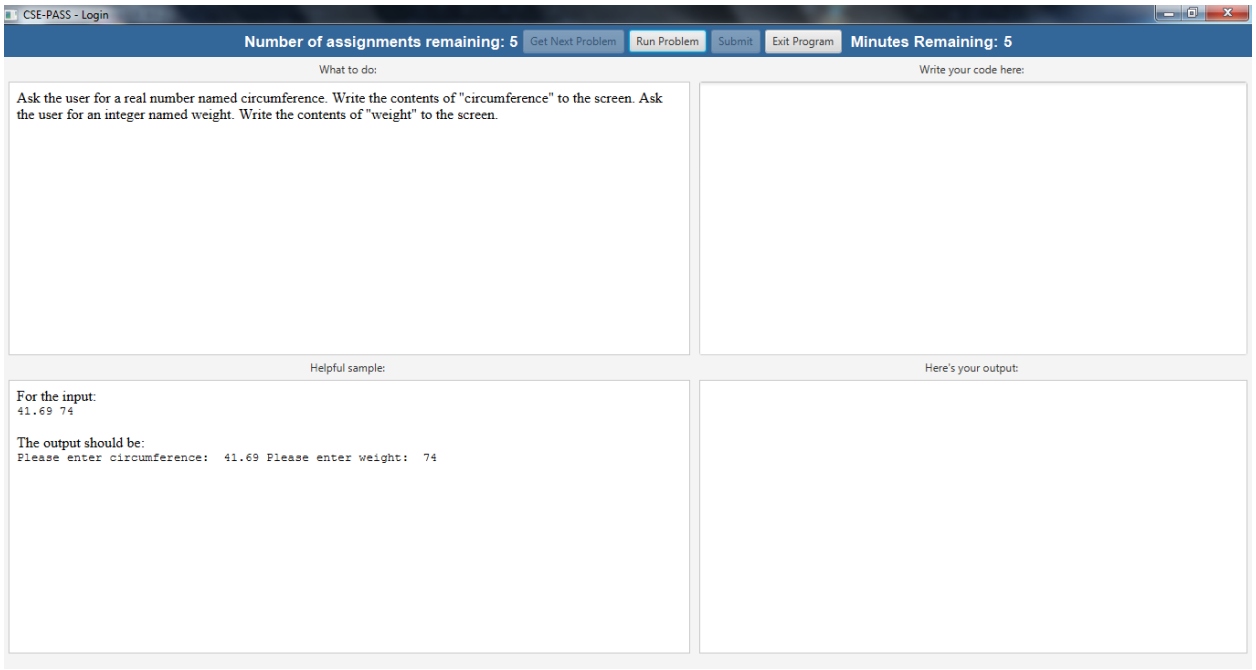


Figure 2: the main application window after retrieving the next problem

Run Problem sends a student's code to the server where it is checked for errors. The server generates an output file which the client side continually pings for until it finds the output file. When the client finds the output file, it is downloaded and displayed to the user.

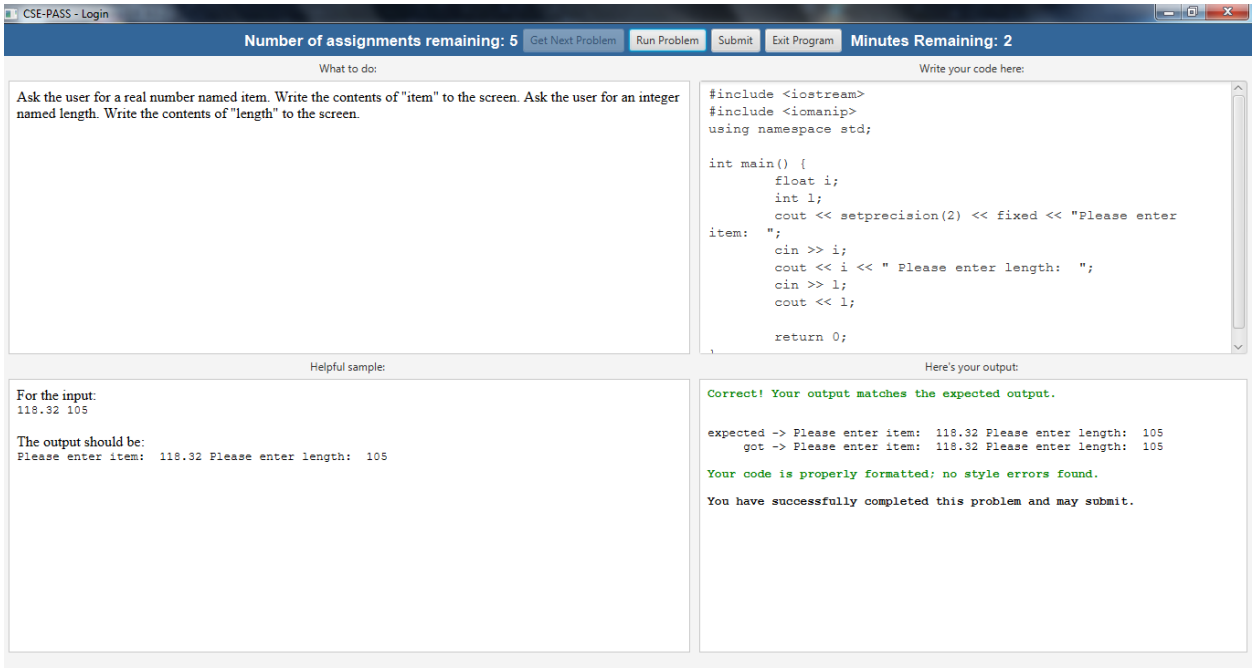


Figure 3: output of running the correct code to solve the given problem

Charles Bevington
Jace Boby
Steven Coraor
Daniel Giannone
Varun Patel
Jonathan Shaub

The Submit Problem button is enabled if and only if the student's code generates the correct output for all test cases and there are no style or formatting errors. When submitted,, the client packages the webcam images with the screenshots and sends them to the server with a copy of your code.

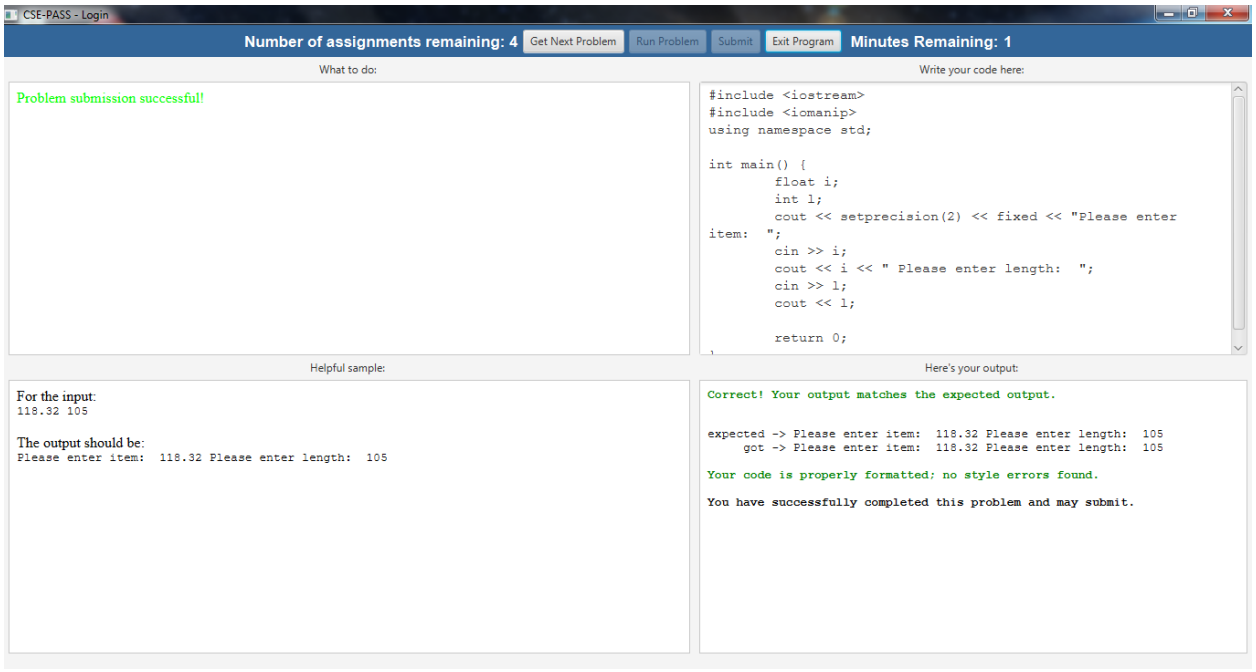


Figure 4: the message displayed once the code has been submitted

One additional security precaution we took was adding cheap php authentication to all of our client-server interactions. When the client wants to talk with the server, it sends all necessary information for processing, but also sends an additional variable called the key. The key is private so that only the server and client know it, and when the client sends it, the server looks for it to confirm it is the key we hard coded in. This ensures, to some degree, that all communications the server responds to, are coming from the client and not a malicious source.

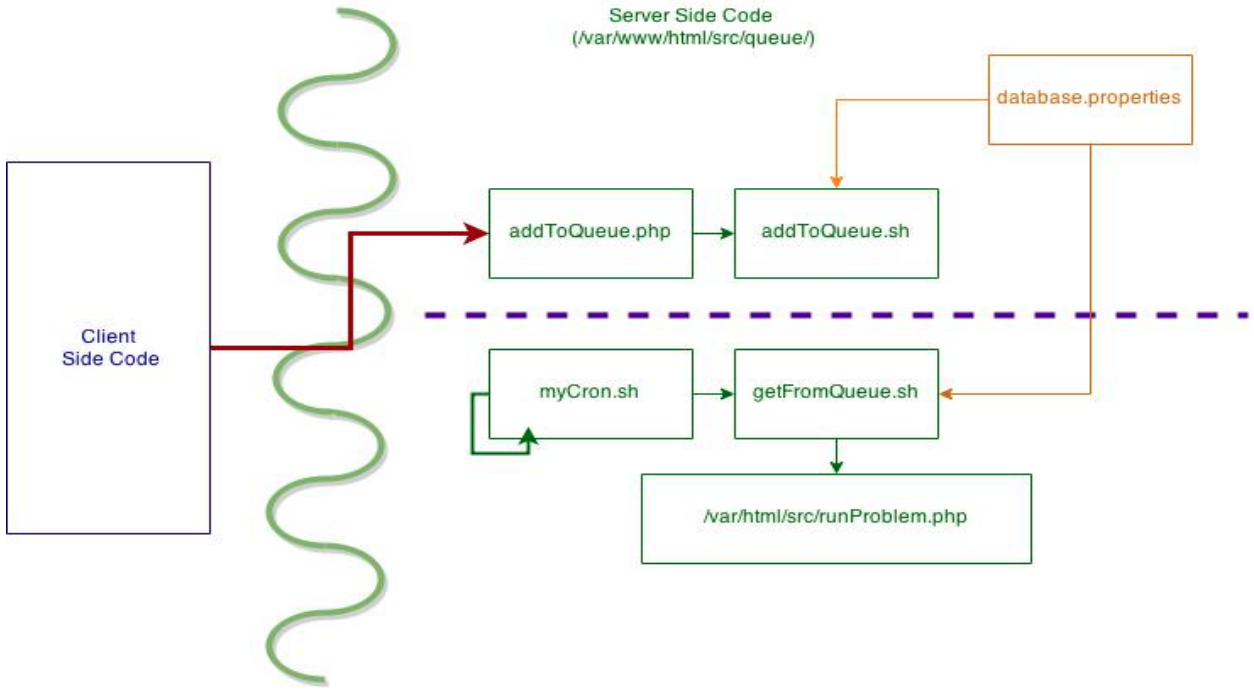
Charles Bevington
Jace Boby
Steven Coraor
Daniel Giannone
Varun Patel
Jonathan Shaub

Queue

The queue was implemented this semester to help with the issue of overloading the server with many workloads in the event that users are working on their assignments at the same time. During the beginning of the semester, the queue was what was assigned to be implemented by the systems team. Different solutions for load balancing were researched by the systems team. One of the solutions was a Amazon Web Services utility that will manage incoming traffic that would hit the server. The downfall of this solution was that there was less control; for all workloads sent to the server, Amazon's utility would "somehow" load balance them and execute/process each workload. This solution was not utilized because the team was unaware of how scalable it is as well as what it would take to reinstall and maintain it in the event that there were several servers or there was a move to another server. Due to this, the systems team stopped considering commercial solutions and began working on implementing their own solution.

At first, the queue was a simple implementation. The client calls addToQueue.php which grabs the workloads and puts them in a unique named directory. There is a byte file that calculates the next prioritized workload on the queue. After some re-designing, the byte file that calculated the next priority workload was deleted and instead a path to the unique named directory workload is appended onto a workload.paths file. Then, another script called getFromQueue.sh will look through the workload.paths file and grab the first (or first "n") paths and delegate control to runProblem.php so that it can perform the workload at those specified directories. There can be many issues with this. For example, when the workloads.paths file is being accessed by addToQueue.php from one client, and another client is trying to access the workloads.paths file there can be issues. To fix this issue, the workloads.paths file is replaced with a database table. The implementation was changed accordingly to accommodate a database table instead of a workloads.paths file.

The following is a dependency diagram illustrating the call stack of the queue.



Charles Bevington
Jace Boby
Steven Coraor
Daniel Giannone
Varun Patel
Jonathan Shaub

Server-side Processing

runProblem.php

When the queue selects the next student submission to run on the server, it sends the relevant information, such as the source code, a timestamp, the username, and the problem ID to runProblem.php. runProblem.php has the following main tasks:

- Parse the problem definition, saving the sample input and expected output for the problem into separate files for checking
- Copy this data into the shafferbox.sh directory
- Call shafferbox.sh to attempt to compile and run the code
- Examine the output files generated from within shafferbox.sh, comparing the produced output with the expected output for each test case
- Generate the results for the running of the problem and save them to a publicly accessible location within the Apache2 server directory.

  Once this output file is generated, the client will be able to download it and display it to the user; until this file is generated, the client will continue to ping the server to check for its existence.

Shafferbox

Shafferbox is a set of scripts that was developed by Jace to replace Sandboxie, which, as described in the Received Status section, was unable to be used due to licensing issues. It is a set of two shell scripts: shafferbox.sh and shafferbox_run.sh.

shafferbox.sh is called by runProblem.php with three arguments: the language of the code to compile, the name of the source file (generated by runProblem), and a directory path containing the sample input files to run the compiled code with. It has the following main tasks:

1. Attempt to compile the code using the compiler defined for the language of the source code. The script was written with extensibility in mind, and it can be modified to add support for other compilers such as Java. Currently, the script can compile C files with the gcc compiler and C++ files with the g++ compiler.
2. Regardless of whether the compile succeeds or not, the next step is running stylecheck on the code to test for style and formatting errors. The results are saved in an HTML file.
3. If the compiler has succeeded in compiling the code, shafferbox.sh copies the executable and the input files to a sandbox directory and invokes shafferbox_run.sh to run it.

   If the compiler has failed, it copies the compiler output file to the folder containing the source code and does nothing more; runProblem.php will find this file and generate the output that the user sees.

   A rudimentary check for fork-bombs is included, but it can be tricked. Currently, it simply checks the source file for existence of the string “fork();”, and if found, it sets the variable containing the compiler’s return status to a value that is specifically looked for. It is then treated as if a compiler error has occurred and generates an output file stating that malicious code was detected. shafferbox.sh will quit at this point and control will return to runProblem.php.
4. When shafferbox_run.sh returns, the generated output files are copied back to the folder containing the source code for runProblem.php to find.
5. The sandbox directory is cleaned of any files related to this particular executable, and the executable itself is deleted. At this point shafferbox.sh will end, returning control to runProblem.php.

The purpose of shafferbox_run.sh is to separate the actual running of the executable from the rest of shafferbox.sh. Thus, the only task performed by shafferbox_run.sh is to run the executable on each of the provided input files and redirect the output streams to files. Currently, the executable is run via a timeout command, which kills the process if it runs too long. As the executable is running in a non-sandboxed environment in this fashion, next semester’s team will need to replace this portion of shafferbox_run.sh with a proper sandbox command, and they should not need to modify anything in shafferbox.sh in order to do this.

Charles Bevington
Jace Boby
Steven Coraor
Daniel Giannone
Varun Patel
Jonathan Shaub

Challenges

There were many challenges that were faced by the team:

- **Learning all of the previously implemented code**- The code that was provided from the last generations of code was not readable. It was hard to determine the set of steps/processes that the application went through to run. One of the biggest issues was to try and follow the code and determine why certain things were implemented the way that they were. This was because there was a lack of comments. To battle this, code was refactored into readable form and cleaned by the team.
- **Relocation to the new server**- Because of VPN issues, the team moved the server side code to an off-site non Penn State server. During the move, there were many things that needed to be copied from the old server to the new server. Determining this was one of the biggest challenges. Only relevant files were copied to the new server and extraneous files were left behind on the new server. Configuring the new server to accept our application was a very big challenge (eg, SSL files, etc.)
- **Removal of "sandboxie," which was an integral part of received code/dummy sandbox "shafferbox"**- The team received the application that was configured with a commercial sandboxie utility. Dr. Shaffer was unaware of this. Removing this utility as well as coming up with a solution for it's replacement was the biggest challenge of the semester. Open source solutions were explored (such as LXC). Ultimately it was decided that a placeholder called shafferbox will be implemented so that the project is able to move forward.
- **Troubles with technology (Git, LXC)**- As described above, LXC/Sandboxie was one of the biggest challenges of the semester. During the beginning of the semester, the team explored the idea of using source control for our code. Because the team was ultimately not successful in using source control with Git, this was abandoned and Google drive along with a changelog.txt file was utilized to upload latest versions of the code for the team to access.
- **Queue system for client submissions/Had to integrate many different pieces of the project**- A queue was implemented to handle workloads. Essentially, the direct communication between clients and the server was broken and instead, the client would only add its workloads to a queue. The server would then look at the queue to process a finite amount of workloads at a time to prevent from overloading and crashing the server. This was a big challenge because when the client was disconnected to the server and both the client and server were connected to the queue instead, there were many challenges that arose (the client not being able to properly communicate with the queue, security challenges, adding cheap PHP authentication, giving runProblem.php the necessary variables it needs to carry out a workload, etc..). For this, the Systems team worked with both the UI team (client side communication) as well as the Backend team (server side runProblem.php) to properly configure each part of the process to that the application is in working state.

Successes

- **Improved Feedback Syntax-** At the start of the semester, there was little to no feedback given to the user regarding what the program was doing. We made it a point over the course of our 14 weeks to make sure the program was giving helpful feedback to the user.
- **Removed old sandbox method-** The students from last semester unfortunately used a commercial software, Sandboxie, for the sandboxed environment in the program. This software required a license, so we were forced to remove any references to sandboxie and create a new sandbox environment. We called this "Shafferbox". It is currently just a placeholder and needs to be implemented next semester.
- **Moved to new reliable server-** The semester began with the project residing on a Penn State server called "Adder". We decided it would be a good choice, for both performance and reliability, to move to a new reliable server. We used a "Server4you" provided to us by Dr. Shaffer. The process of moving all of the projects components to the new server took roughly three weeks.
- **Added cheap PHP authentication-** Cheap php authentication was added in order to provide a quick and easy boost to security.

- **Queue system-** When it comes to distance education, often times students tend to wait until the last minute to submit their code. This would cause issues if the system wasn't able to handle a large influx of student submissions toward the deadline. In order to deal with this, we implemented a queue system that helped modularize the submission process.
- **Reworked login screen UI-** The login screen user interface was updated to provide more feedback and also look a bit cleaner. It now displays proper error messages, and also has a check box to make sure the user accepts having their screenshots and webcam photos taken.
- **New Look -** Various changes were made to the UI over the course of the semester. On the main screen, we adjusted the size of the different fields/boxes in order to better accommodate the users code. Various formatting improvements were also made in order to make the problem/examples easier to read.

Lessons Learned

One of the most important lessons we learned this semester while working on CSE-PASS stemmed from the trouble that we encountered when looking for an alternate sandboxing method. There was a period of two or three weeks where very little work was done while we focused on trying to find some way to create a sandbox on the server. Eventually, Dr. Shaffer suggested that we shelve the sandbox feature and tackle it at a later date, so we abstracted the sandbox into what we called "shafferbox," which handles everything that the sandbox will need to do except for the actual sandboxing procedure. From this ordeal, we learned that sometimes it is better to put a feature on the backburner if trouble is encountered and move onto and work on another portion of the project.

The second lesson that was picked up this semester was one that Dr. Shaffer made sure was ingrained in our minds: Testing is crucial. The testing process involves going through every possible path in the program to determine if it meets the required specifications given by the customer. Dr. Shaffer made sure that we understood that a program's correctness cannot be proven through testing; rather, testing is a way to make sure that as many features as possible work as intended. Bugs and errors will always be found upon the launch of the software, but testing can reduce this number. Unfortunately, we did not focus on testing this semester; instead, we only fixed bugs that appeared to us. Therefore, the next group who inherits this project will have to put it through rigorous testing.

Phase 3 Requirements

Although we accomplished a fair amount of work this semester, there is still plenty to be done when the next group arrives. For one, the software must be tested, as previously mentioned. Also, a proper sandboxing method will have to be implemented on the server to ensure that user code is compiled and run in a safe environment. Another security issue that will need to be addressed pertains to webcam photos and screenshots taken by the software. These are saved locally on the user's machine, but are entirely unprotected. This means that the user has complete access to the photos and screenshots and can modify, delete, or add to them with ease. This opens up possibilities for cheating, which must be avoided due to the academic purpose of the software.

These are all critical features that need to be addresses as soon as possible since they deal directly with software security, however, there are smaller fixes that need to be done. For example, the user interface on the main problem screen should be reworked. Currently, the sizes of the four text boxes are fixed and this leads to them having varying sizes depending on the size and resolution of the user's monitor. In order to give each user the same experience, the text boxes need to be redone so they appear the same, no matter what computer it is running on. The "Time Remaining" timer, which lets the user know how much time is left until the current problem can no longer be submitted, only displays minutes as of now. The timer should also display seconds so the user has a much more accurate representation of how much time remains.

One of the biggest features that has yet to be implemented is some sort of admin utility. This would allow instructors to add/remove students from a course, upload problem sets to the server, and receive submitted code for grading. This feature was not discussed at all this semester and it is entirely up to Dr. Shaffer at this point on when he wants this feature included.

Charles Bevington
Jace Boby
Steven Coraor
Daniel Giannone
Varun Patel
Jonathan Shaub

Summary

At the beginning of the semester, we received the CSE-PASS software in a state where many of the features were in working condition, such as logging in and receiving new problems from the server. However, some features, such as submitting finished code, were fixed by our group. We updated the login screen so it looks much more clean than it did before. Overall security was increased through PHP authentication and a queue system was implemented so the server can handle heavy loads of submissions and “Run Problem” requests. We also took the time to organize the source code on the server so the next group will know exactly where to look for the files. The previous sandboxing method was removed due to it being commercial software and shafferbox was created and implemented. There is still a lot of work to be done, which will be the next group’s task. They will have to find an appropriate way to sandbox submitted code on the server and update the UI of the problem screen. Overall, this course did a great job at teaching us what a true development environment is like and how to successfully work as a team to accomplish the specifications set by the customer.