CONTAINERIZATION OF AN AUTOGRADER

by

Jonathan R. Martin

November, 2023

Director of Project: Nic Herndon, PhD

Major Department: Computer Science

Evaluation of programming assignments is a critical component to many computer science courses. The grading process of these assignments is often an arduous task due to the complexity of the technologies used in each assignment. Educators have addressed the issue by developing tools that automate the grading process of programming assignments. These tools are able to evaluate hundreds of student submissions in a matter of seconds in a consistent manner. However, a common drawback shared by many of these autograding tools is the steep learning curve required to install and operate the tools. Furthermore, providing students with a self-evaluation tool is not possible with many of the autograders that are available. This project provides a solution to these shortcomings by utilizing containerization. The open-source tool AutoGrader has been containerized using Docker. The AutoGrader container allows instructors and students to grade assignments in any environment that has Docker installed. A results aggregation component was also added to the container to streamline the uploading of grades to a learning management system. Documentation of the project along with a detailed tutorial has been composed using GitHub.io. This documentation will help users learn the system quickly and provides vital information for individuals wishing to customize this project for their own work.

Containerization of an Autograder

A Project

Presented to the Faculty of the Department of Computer Science

East Carolina University

In Partial Fulfillment of the Requirements for the Degree

Master of Science in Software Engineering

By

Jonathan R. Martin

November, 2023

Containerization of an Autograder

By

Jonathan R. Martin

APPROVED BY:

DIRECTOR OF PROJECT:	Nic Herndon, PhD
COMMITTEE MEMBER:	Venkat Guidivada, PhD
COMMITTEE MEMBER:	Rui Wu, PhD

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The need to assess students learning has always been a fundamental component of the education process. Homework assignments are a common way that this can be accomplished. The field of computer science adds some unique challenges to this process due to the complex media used in their assignments. These assignments often involve programming in which students must submit running code in the form of files. These are complex files that can be assessed in many different ways. It is the instructor's responsibility to evaluate these submission files. Often, this involved an instructor manually downloading each student submission and running these files on their local machine. An instructor may have developed test cases that could be run to check the validity of a submission. This speeds up the process, but there are still many manual steps for the instructor to perform. Over time, educators have developed tools to automate the process of grading programming assignments.

Tools designed to provide automatic evaluations of student assignment submissions are called automatic graders or autograder for short. The primary goal of an autograder is to automate the grading process. There are numerous paid and open-source autograder tools available on the market. The features provided by each tool vary widely, but there are several commonalities that can be observed amongst these tools. Autograder tools can automatically assess large quantities of assignment submissions quickly and consistently using predefined grading criteria. Many auto graders are also capable of providing feedback for students by highlighting errors or listing the

incorrect answers. Autograder tools come in many different formats and the runtime environment ranges from simple command line programs to cloud-based applications accessed via a web browser. Many autograder tools are complex to use and require extensive setup and training in order for an instructor to become efficient with the tool.

The primary benefactor of these tools are the course instructors. Automatic grading tools enable the instructor to quickly grade assignments. Additionally, assignments are graded consistently and objectively [7]. These tools are vital for large class sizes. Students also benefit from these tools. Assignments are graded without bias, and it is possible to provide immediate feedback to the students upon submission. Students can use this feedback to improve their solution before resubmitting an assignment. Autograder tools bring many benefits to the classroom which has led to wide spread adoption of these tools.

A survey of available autograder tools was conducted to establish common functionality among these tools. There were several drawbacks identified frequently with these tools. All the tools that were reviewed allow their users to automatically batch grade student submissions. These tools also automatically generate some form of output that contains grading information. Autograder functionality varied widely beyond these basic functions. Each tool has its own unique set of drawbacks such as limited grading capabilities and overly complex solution creation processes. In addition to unique issues, a common short coming of autograders is the steep learning curve required when learning how to use the tool. Setting up the correct environment can be

cumbersome for many of the autograders. Providing students with a way to assess their work prior to submission is not even possible with some of the tools.

This report discusses a solution to the issues commonly faced while using other autograders. The main issue addressed was reducing the initial cost (time and effort) required to use an autograder. This was accomplished through the containerization of an autograder. Containerization is the process of packaging a piece of software and all of its dependencies into a lightweight executable. This executable is referred to as a container. Containerization is popular, because it makes the process of creating and deploying applications faster and easier [6]. Deploying the autograder tool inside a container makes it easy for an instructor to download and run. The arduous setup process has been eliminated by using containerization. This benefit extends to each use of the autograder. Instructors can use the tool quickly each time it is deployed for an assignment. This convenience also makes it feasible to distribute limited versions of the containers to students for self checks. Students can run the containers in their unique environment without complications.

This project had additional goals beyond the containerization of the autograder tool. These goals align with the overall goal of improving the ease of use of an autograder. The first of these goals was to create an output file summarizing the grading results of all submissions graded. The system was only capable of producing individual results files for each submission at the beginning of the project. The autograder was modified to automatically produce a single file that contains all the information vital to an instructor in regards to the assignment. This single output

file was designed to output a format that is compatible with popular learning management systems (LMS) in order to permit an instructor to upload this grade file to the LMS being utilized for the course. Next, the process of running the container was streamlined. This was accomplished by designing the container to perform all of its required tasks by running a single command. Also, a detailed user manual was created to improve onboarding of new users. These improvements made the containerized autograder easier to use and more efficient for instructors.

Chapter 2

Autograding Tools

A survey of available autograding tools was conducted to establish common functionality of this tool type. There are numerous tools that provide some degree of autograding capabilities. This survey is not exhaustive as the tools discussed were limited to more popular autograders. The numerous tools available to instructors can be grouped into a paid category and a free category. Documentation of which features were shared between these two categories and what features were exclusive to the paid options was made in this report. Additionally, the drawbacks of each tool were evaluated. Summaries of each tool analyzed during the survey is provided in this section.

CodeGrade

CodeGrade is a popular paid solution for automatic grading of coding assignments. The tool started out as a project to make grading easier for teaching assistants at the University of Amsterdam. CodeGrade is now a professional quality software product that is used by numerous universities across the world. The tool works with over 180 different programming languages, and it is compatible with frameworks like Flask for Python-based web development. CodeGrade provides users with a clean user interface that is easy to use. There are built-in test types for common testing situations such as input/output tests. Using these built-in tests reduces the setup time required by an instructor to create tests for an assignment. CodeGrade has several

purchasing options including individual student payments, institutional licenses, and enterprise options [2].

CodeGrade provides an online integrated development environment (IDE) where students can write and test their code. Use of this IDE is optional. Instead, students can code in their preferred environment and upload files to CodeGrade or students can use the CodeGrade GitHub integration. Using this integration enables feedback for every git push executed by the student. CodeGrade includes other valuable features such as plagiarism detection and integration with learning management systems such as Blackboard. Endorsements by users of CodeGrade highlight the intuitive user interface and the ability to easily integrate CodeGrade with their current technology. An additional benefit of this paid option is that there is a dedicated team to solve technical issues that may arise. In contrast, the troubleshooting process for open-source options can be more arduous.

codePost

Another popular auto grading tool is codePost. This tool is free for educators, but enterprise uses require a payment. At its core, codePost is an autograder. However, there are several features in codePost that make it stand out from similar tools. Instructors can review graded submissions, and add inline comments in the students code. The comments will appear adjacent to the specific line of code to make it easier to read. CodePost has several other annotation features that make it easy to leave valuable feedback for students. Another powerful feature is the ability for students to provide feedback to instructor comments, so the instructor can gauge the effectiveness of their

feedback. Other valuable features in codePost include plagiarism detection, analytics, and integration with other systems via the codePost API [4].

The basic workflow of codePost begins by creating an assignment. This requires only a few clicks and providing a name for the assignment. Next, an instructor can begin composing tests to be run automatically. CodePost provides the ability to upload an existing test script or the option to write new unit tests using the tool. A solution file for the assignment can also be uploaded which allows the system to check that all tests are passing when given the correct solutions. Once the tests are ready, the instructor can make the assignment submission available to students. Restrictions can be placed on the submission upload files such as the file name and type. From the student perspective, the user interface is very simple. Upon navigating to the console page for a specific course, students will see all assignments that are available for submission. A button labeled “Upload Files” is displayed along with additional details about the assignment. Instructors can also choose to have students submit assignments via a learning management system or GitHub and import the submissions to codePost. The tool will automatically run the tests upon submission. Manual grading is also available during which instructors can choose to leave feedback on an assignment. After all of the assignment submissions have been graded, instructors can view statistics about the assignment. Once the instructor is satisfied with the assignment grading process, they can publish the results to make it visible to the students.

Codio

Codio provides users with a cloud-based platform for computer science education. This platform aims to create an environment where instructors can create and teach online computer science courses. Codio is a complete learning environment and not just an assignment grading tool. Instructors can create interactive lessons, automated assessments, and more. The Codio platform is a browser-based system and designed to scale. It also has an online IDE which makes it easy for students to practice and complete assignments. Other useful features that are part of the Codio platform include a code plagiarism checker, integration with learning management platforms, and templates for common course resources. All of these resources make it easier for instructors to create and manage their courses. There are several pricing models depending on use-cases. This system is used by academic institutions and industry [3].

Codio provides manual and automatic grading features. The grading system is divided into two parts. There is a standard grading form that can handle standard input situations and typical unit tests. The standard grading form is easy to use and handles most situations. There is also an advanced grading form that allows the instructor to write custom grading scripts in any language. Codio is capable of grading a variety of assignment types including multiple choice tests, fill in the blanks, free response and coding assignments. This is useful for courses that have many assignment types. Codio provides many of the features found in other autograder tools while also providing a more comprehensive learning environment.

Otter Grader

Otter Grader is an autograder tool developed by UC Berkeley. This is an open-source tool that is capable of grading Python and R assignments in the form of executables, Jupyter Notebooks, and Markdown files. The workflow allows instructors to customize the tool to meet their needs. Otter Grader is elegantly designed to abstract from the internal tasks associated with an autograder which makes the tool more accessible to instructors from various disciplines. A key advantage of this tool is that it can run in many environments. Otter Grader can be run on a local computer, a server, and JupyterHub. The tool is also compatible with learning management systems such as Gradescope and Canvas. It is also possible for instructors to provide a client package to students that allows them to run checks on their assignments prior to submission [9].

Otter Grader can be divided into six parts: Otter Assign, Otter Generate, Otter Check, Otter Export, Otter Run, and Otter Grade. Each of these areas represents a different part of the grading process. The Otter Assign component is where an instructor can create an assignment. Instructors will compose questions and solutions in the form of a notebook and Otter Grader will convert this information as needed. Otter Generate creates the necessary configuration files for the autograder tool to run. The processes contained in the Otter Generate component are done automatically when using the Otter Assign component. Students are able to run tests that were made available by the instructor using the Otter Check component. This is a command line tool that students can run to check their solutions. The Otter Export component allows instructors to generate PDFs of the notebooks which can be useful for manual grading as well as creating a solutions PDF. Otter Run allows assignments to be graded locally without containerization.

However, non-containerized grading is not as secure. Instead, the Otter Grade component is recommended. This component handles the grading of assignment on a local computer using Docker containers running in parallel. The result of this process is a CSV file containing the grading summary.

OK

OK is an open-source suite of software that provides instructors with tools for automatically evaluating programming assignments. The software was initially developed at UC Berkeley to handle the grading of programming assignments for a class with over one thousand students enrolled. OK has continued to be adopted by other computer science courses at UC Berkeley and other institutions. The software uses the data generated during the grading process to produce statistics for the assignments. This information can enable instructors to discover insights such as the effectiveness of a given assignment. OK is also capable of plagiarism detection and integration with learning management systems [8].

There are three parts of the OK software suite: a client, a server, and the autograder. Each of these parts can be used independently or together with the other parts. The OK client enables students to perform local testing on their assignments prior to submission. Students can connect to the OK server while using the OK client to allow for the submission of their assignments. However, it is optional to connect with the server during this process. Next, the OK server is where students submit their assignments for grading. The OK server also provides a dashboard for instructors to view and control assignments. Using the OK server is free for instructors of

courses that meet specific criteria. The autograder component of the system is designed to run using the OK server. However, the OK autograder can be setup to run on any server.

Web-CAT

Web-based Center for Automated Testing (Web-CAT) is an open-source automatic grading system. The tool can grade programming assignments in several languages including Java and C++. The design of Web-CAT makes it very customizable, so the tool can be applied to other languages. This flexibility allows Web-CAT to handle many different types of assignments. This includes full-scale programming assignments and not just small functions. Web-CAT is also capable of grading student-written tests. It can measure the code coverage achieved by the student-written tests and grade test thoroughness. The evaluation of student-written tests by the tool is uncommon among auto graders. The ability to grade student-written tests allows instructors to encourage students to develop quality tests for their programs [Web-CAT]. The robust capabilities of Web-CAT have made the tool the most used open-source automatic grading tool for programming assignments [5].

Web-CAT is a web-based system, so all of its features can be assessed via a web browser. This includes the student submission process and viewing results of graded assignments which makes this tool easily accessible to the students. The administrative tasks performed by the instructor are also accomplished using the web interface. The system architecture has also been designed to accommodate plug-ins. An instructor can develop their own plug-ins to handle their specific situation such as a unique grading scheme. These plug-ins can be uploaded for use without

impacting the underlying system that is running on the server. Web-CAT has a very active community of contributors which make it easy for instructors to find solutions to their issues. The design of Web-CAT make it a portable and secure system while permitting extensive customization [13].

Another open source autograder identified during the survey is AutoGrader. AutoGrader is a powerful tool for automatically grading programming assignments. This tool was selected to be used in this project and will be discussed in more detail throughout the report.

At the conclusion of the survey, a clear divide in features appeared between the paid and free tools. In general, the paid tools provided a superior user experience. This came in the form of well-designed user interfaces. Also, the paid options require minimal effort to get started grading. Paid options provide capabilities to easily import final grades into a learning management system, and some of the paid options were so feature rich that the tool can serve as a learning management system. In contrast, the free options require extensive setup and are complicated to use. In general, all of the tools surveyed provide basic automatic grading of programming assignments. However, some free options are only capable of grading specific programming languages. All of this information contributed to the development of the goals for this project. Providing an open-source autograding tool that is easy to set up, capable of grading multiple languages, and easy to export grades became key goals for this project.

Chapter 3

Design & Implementation

The design of the containerized autograder began during the research phase of the project. While reviewing existing autograder tools, the common shortcomings of these tools were documented. There were many reoccurring issues noted that could be classified as user interaction issues. The setup process was very complex for several tools. Many of the user friendly options are locked behind a pay wall. These observations played a critical role in shaping this project. The overall goal for the design phase was to create an easy to use autograder tool. Over the course of the design phase, this goal evolved into the final product discussed in this paper. This chapter of the report discusses the design process and key moments during the design phase.

Selection of Autograding Tool

A review of available open-source autograder tools was performed to identify a tool to be integrated with the project. Several tools were considered and there were several considerations made during the review of each tool. Here are several of the most critical criteria considered during the selection process; is the source-code publicly available, can the tool assess multiple programming languages, is the tool regularly maintained. Many of the tools focus on one language, so they were eliminated. It was also difficult to find a tool that provided the complete source code. This was seen as crucial as it is important to understand how the tool works before making modifications. The open-source tool AutoGrader was selected. The source code for this

tool is available at the following URL: https://github.com/zmievsa/autograder. In addition to having publicly available source code, the tool is capable of grading C, C++, Java, and Python. It is also possible to grade any programming language in a limited capacity in which the system only checks standard output. The repository for this code has recent commits during the year 2023 in addition to the previous several years. AutoGrader was deemed as the best fit for this project. This tool was integrated into the system to create the containerized autograder tool.

AutoGrader provides an extensive list of features to its users. It provides plagiarism detection methods. Grading rules such as points totals can be customized. Writing test cases for an assignment is a streamlined process. These are only a few of the many features provided by AutoGrader. The tool makes an effort to streamline the process of automatic grading of programming assignments. This project will build off of this process and provide an even easier to use tool. In addition to improving the tool, it was important to avoid damaging the existing tool. The functionality of all of these features was preserved during the containerization process.

The tool can be executed with a run command in the directory containing student submissions for the assignment. The AutoGrader tool requires a specific directory structure for the tool to work correctly. A main grading directory must be created. All of the student submissions should be placed in this directory. Another directory named “tests” must also be placed inside of the main grading directory. The tests directory contains several other directories; extra, input, output, test cases. There is also a config file that can be used to modify the grading process such as maximum allowed run time for student submissions and assigning grade weights to test cases. A

standard output formatter file is also provided. This file contains functions that can be used to format output from the student's submission to allow for accurate comparison to the test case output. The directory structure discussed is depicted in Figure 1.

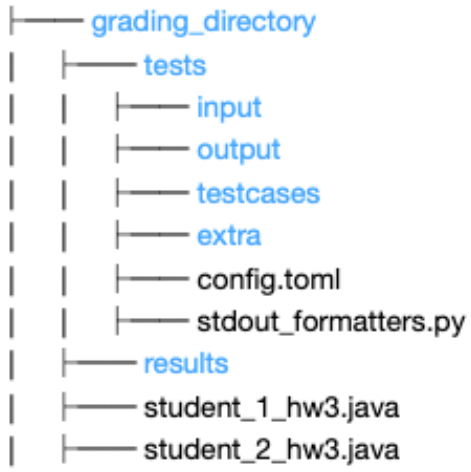


Figure 1 - Directory Structure required to use AutoGrader. Folders are shown in blue. Files are shown in black.

Test cases, input, and output check files must be written by the instructor and placed into their respective folders. AutoGrader will check each directory during execution. Upon execution of the run command, a results folder will be generated inside the main grading directory. This folder contains a results file for each student submission that was graded. Each results file contains a list of test cases that were checked along with their outcomes. An overall grade for the assignment is also provided. Additional comments may also be provided if errors were encountered while grading. This grading structure was preserved by this project to ensure that all features of AutoGrader remained functional with the container. Zip files containing the folder structure for a Mac/Linux version and Windows version is provided at the following URL: https://github.com/jmart5/containerized_autograder.

Containers

The decision to run the autograder tool inside a container was made to provide easy access to the grading tool. Containers are the ideal method for accomplishing this goal. In order to understand why a container is a good solution for this project, it is important to establish how a container works. The name container is used to describe a unit of software. This unit contains the code for an application and all of the dependencies that are required to run the application. Everything needed to run a specific application is inside its container. Since everything required is provided inside the container, it is easy to run containers in any environment. This is one way that containers will make the AutoGrader tool easier to use. An alternative method is to run the autograder inside a virtual machine. This would provide the same benefit of providing all of the required dependencies inside the pre-made environment. However, virtual machines are much larger pieces of software. A container is a lightweight executable application that uses resources more efficiently that a virtual machine. Using a virtual machine to host the autograder tool would be an inefficient use of resources, thus the container is a better option.

Containers are a generic concept with several implementations to choose from. Docker has been selected as the specific tool used for creating and running containers for this project. Docker is more than just containers. It refers to the platform and tools it provides for creating, running, and managing containers. The tool can be run from the command line or using the Docker Desktop application. Docker is popular because it enables developers to quickly test and deploy their software [11]. This included deployment into a production environment. Images are a key part of this ecosystem. An image contains everything needed to run an application. It can be thought of

as a template that is used to create containers. There are many base images available for customization and it is also possible to create an image from scratch using a Dockerfile. An image is similar to the concept of a class in object-oriented programming. An instance of an image is a container. There can be multiple instances of an image similar to classes and objects. The container is where the software is running. The Docker Engine is what powers the Docker platform. It creates and runs containers. Docker Engine is a layer located between the host operating system and the location where the application is being virtualized. This is different from a virtual machine which utilizes a Hypervisor. The lack of a Hypervisor leads to a significant improvement in performance of Docker compared to a virtual machine [10].

The process of running a container from an existing image is made easy by Docker. Containers can be run in any environment where Docker is installed and running. Users only need to verify that Docker is running on their system and that they have downloaded the desired image. The creation of an image that contains AutoGrader and the necessary dependencies will make it easy for users to gain access to the autograding tool. This design overcomes the setup issues encountered by several other tools. The installation process of the tool Otter-Grader is one example of setup issues. First, the tool was installed using a command in terminal. There were several necessary libraries that needed to be installed in order to use the tool. If anything was missing, the user would receive a lengthy error message. Once all missing libraries were installed, the Otter Assign command could be executed. This is a command that generates necessary files for grading. After this step, the run command could be executed to grade the assignment. However, additional dependency errors were encountered. Overall, the process

required to make that tool work was slow and frustrating. This is one example of a setup issue. Other tools had similar issues when they were investigated. Running the grading tool inside the container eliminates these issues.

Building the Autograding Image

An image containing the AutoGrader tool and its dependencies was built using an existing base image. Creating a new image by modifying a base image is a common technique. This is an efficient approach that allows developers to take advantage of a pre-existing image that already contains common software packages that are likely required. The alternative to this approach is to create a Dockerfile. This is a text-based script that provides instructions for building a Docker image. Composing a Dockerfile is time-consuming and it is a very manual process. It is also error prone and can be resource intensive. These disadvantages prompted the use of the first approach. Next, a base image was selected for the starting point. The latest version of the Ubuntu image was selected. This is the official Ubuntu image provided on Docker Hub. Canonical, the organization that produces Ubuntu, actively maintains this image. This official support made the image attractive. Also, Ubuntu is a familiar Linux environment which made development easier.

Once the base image was selected, construction on the new image could begin. The first step was to use the ‘docker pull’ command to pull the latest version of the Ubuntu image. Next, the run command can be used to start a container using the Ubuntu image. The ‘-it’ option should be used to start an interactive session with the container. This allows the user to interact with the Ubuntu command line inside the container. This image has minimal packages installed, so

Python and ‘pip’ needed to be installed via the command line. Next, the AutoGrader tool was installed using the ‘pip’ command. A new directory titled ‘grading_dir’ was made to provide a designated location to perform auto grading tasks. Inside this directory, the necessary configurations and directories were created. This directory also serves as the location to place all submission files for grading. Now, the image contains the necessary test directories. The container setup was completed and the container exited.

Once the container was setup with the desired installs, an image was created from this container. The container needs to be running to begin this process. This can be verified by viewing the list of active containers. The ‘docker commit’ command can be used to create an image from the running container. After using this command, a new image is created that contains all of the necessary resources to run the AutoGrader tool inside the container. Figure 2 lists the commands used to create the new image.

```
Step 1: Start a container:
% docker pull ubuntu
% docker run -it ubuntu

Step 2: Setup dependencies inside the container
% apt-get update
% apt-get install -y python3 python3-pip
% pip install autograder
% mkdir grading_dir
% autograder guide
% exit

Step 3: Create a new image:
% docker commit <container id> <new image name>
```

Figure 2 - Docker Image Building Steps. The first two commands can be used to start a Ubuntu container. The next group of commands setups up dependencies inside the container. The last command creates a new image from this container.

The image building process was performed several times during the project. Each time new components were added to the system, a new image needed to be built. This required additional steps to those previously discussed. For example, the development of the CSV results output component required a new image to be built. A Python script file was copied into the running container. This script had the pandas Python library as a dependency, so that was installed inside the container. Once all of the necessary components were added, a new image was created. The final Docker image is available on Docker Hub at the following URL: https://hub.docker.com/repository/docker/jmart5/containerized_autograding/general.

The AutoGrader tool requires a specific folder structure in order to run. This folder structure exists inside the container under the directory “grading_dir”. Part of the grading process requires the instructor to place all of the student submissions inside this directory. It also needs all of the test case files placed in their respective directories. This raised the question of how to efficiently provide these files to a running container. The first option explored was copying the files from the local machine into the running container. The command “docker cp” can be used to accomplish this task. This command requires the path to the files on the local machine that are to be copied in addition to the destination path inside the container. Users of the system would be required to manually execute the copy command on all required files for a grading session in order to perform the auto grading. This task would quickly ruin the user experience. The copy approach was used briefly during testing, but was deemed too manual for final users. Ultimately, this option was abandoned in favor of mapping directories from the local machine to the running container.

Mapping directories is the process of associating directories from the local machine to directories inside the running Docker container. This is a common approach used for exchanging data between the local machine and the container. The term volume is often associated with this process as a mapped directory is called a volume. The “-v’” option must be added to the “docker run” command upon startup of the container. A sample of the mapping command is shown in Figure 3.

```
docker run –v "$(pwd)":/grading_dir grader_container
```

Figure 3 - Docker Mapping Option. The current directory on the local machine is mapped to grading_dir upon startup of the container made from the grader_container image.

The directory mapping example shown in Figure 3 only maps a single directory. The AutoGrader tool has several directories that contain various test case files. All of these directories also need to be mapped in order for the AutoGrader tool to run in the container. The solution was to create an identical directory structure on the local machine to match the directory structure created inside the container. This directory structure is shown in Figure 1. A zip file containing this directory structure can be downloaded from the following URL: https://github.com/jmart5/containerized_autograder. Users of this system are able to download the premade directory structure for use while grading. Also, providing a specific directory structure ensures that the mapping options will remain unchanged. Mapping options were provided for each directory in the provided directory structure. Users simply need to add student submission files and test case files to their respective folder inside the premade directory structure and the system will automatically map each folder.

Developing a Component to Automatically Generate CSV Results

The overarching goal of this project was to make it easier to use an autograder tool. Containerization of the AutoGrader tool helps to reduce the setup and execution tasks. Another area that could be improved was the grade outputs. Currently, the AutoGrader tool produces a results output file for each student submission. This output file lists the status of each test case that was performed. A final grade for the entire submission is also provided. In this current situation, an instructor would need to manually retrieve the grade and comments from each file to input the information into a learning management system such as Canvas. The development of a CSV results component was done to automate this process. This component was designed with the goal of automatically parsing the comments and final grades from each results file generated by the AutoGrader tool. Then the CSV results component aggregates all of this information into a single CSV file. The format of this CSV file is compatible with the format specified by popular learning management tools. The addition of this component allows instructors to simply upload the final aggregated results CSV file to the learning management system, thus further streamlining the grading process.

Two approaches were considered for the architecture of the CSV results component. The first option was to develop a component that is independent from the AutoGrader tool. The component can be installed inside the image. Then it is executed upon completion of the AutoGrader run command inside the container. The second approach places the CSV results generation code directly inside the AutoGrader source code. This is a more integrated approach that would not require parsing to retrieve information. However, the first approach was selected

due to its independence from the AutoGrader tool. In this situation, separation was viewed as an advantage. If future changes are made to the AutoGrader tool, the CSV results component can be easily updated independently to remain functional. The component only needs to know the file structure of the results that are produced by the AutoGrader tool. The independent approach will allow the tools developed in this project to be easy to adapt as needed in the future.

An isolated environment was created for the development of the CSV results component. Isolation from the AutoGrader tool and the Docker container simplified the development process of this component. Development was focused on creating a working component prior to integration with the overall system. Creation of this isolated environment was accomplished by creating a new directory that contained only sample output file generated by the AutoGrader tool and the Python file developed for this component. This environment was used for the entire development of the CSV results component, and upon completion the component files were integrated with the system as a whole.

The task of developing a component to automatically generate a CSV output file was deconstructed into several steps. The first step was researching techniques that can be used to generate a CSV file in Python. The Python programming language was selected, because this is the language that the AutoGrader tool was primarily developed in. Using the same language as the AutoGrader tool reduces the new dependencies that need to be installed in the Docker container. The first option for CSV generation is to manually create and format the data. This requires the use of the open function that can then be written to a CSV file. This option was

quickly eliminated, because it lacks the features of the other options and is overly complex for this task. The next option was to use the ‘csv’ Python module. This module is built into Python, so there is no need to install anything. This module is a step up from the manual option, but it is too simple for this purpose. This task requires manipulation of data such as filtering and grouping. There is no easy way to achieve these steps with the ‘csv’ module. Instead, the ‘pandas’ Python module has been selected for the generation of the results CSV file. Pandas provides several useful data manipulation functions that are easy to use. Many of the tasks that could be achieved with the ‘csv’ module can be done in a more concise manner in ‘pandas’.

A results file is generated for each submission by default in AutoGrader. The goal of this component is to extract the desired information from each of these files and aggregate the information into a singe CSV file. Two Python modules were used to accomplish this task. First, the ‘os’ module was used to handle file manipulation. The walk function of this module was used to iterate through all of the results files generated by the AutoGrader tool. The ‘re’ module was also vital to this task. This module provides tools for handling regular expressions. Multiple functions were written that utilized the regular expression tools to identify patterns that indicated the comments section and final grade section in the results file. Student names were also extracted from the files. The results file names are based on the homework submission names, so a file name format was created. This ensures a consistent file naming pattern that can be parsed using regular expressions. The submission file name format is shown in Figure 4. It is critical that each submission is correctly named, so the parsing functions extract the correct name for each student.

```
lastName_firstName_hw#.fileExtension

Example: Martin_Steve_hw7.c
```

Figure 4 - Student Submission File Naming Convention.

The CSV results component contains several parsing functions. These functions are used to extract the desired information from each student results file. This includes parsing functions for the student names, grading comments, and final grades. The name parsing function is shown in Figure 5. This function parses the file names of each results file in the directory. The os.walk function is used to traverse the directory. Each file name is spliced using the underscore symbol as a delimiter. The names of each file are saved into the names list. The grading comments and final grades parsing functions work in a similar manner. Once all of the necessary information was parsed from a student results file, the information was added to a data frame. This process is repeated until all of the results files have been parsed.

```
def parse_names(directory):
    names = []
    for root, dirs, files in os.walk(directory):
        for file in files:
            file_name, _ = os.path.splitext(file)
            parts = re.split(r'_', file_name)
            if len(parts) >= 2:
                part1 = parts[0]
                part2 = parts[1]
                names.append([part1, part2])
    return names
```

Figure 5 - Name Parsing Function. The file name of each submission results file is parsed using this function to extract student names.

After parsing every results file, the complete data contained inside the data frame is exported to a CSV file. The format of this CSV file is based on common fields required by learning

management systems. These fields are student last name, student first name, assignment grade, and comments. The data frame was created using pandas and manipulated to organize the parsed information into the desired format. The data frame structure has a function to create a CSV file that was used to create the CSV results file. A sample of the CSV results component output is shown it Table 1.

Last name	First name	Grades	Comments	
Washington	Elizabeth	80	q1_test1.py q2_test1.py q2_test2.py q3_test1.py q4_test1.py	100/100 0/100 (Wrong output) 100/100 100/100 100/100
Marks	Steve	60	q1_test1.py q2_test1.py q2_test2.py q3_test1.py q4_test1.py	100/100 0/100 (Wrong output) 0/100 (Wrong output) 100/100 100/100

Table 1 - Sample CSV Results Output

Upon completion of the results CSV generation component, the next task was to integrate this component into the system. Since the component was designed to run separately from the AutoGrader tool, the component can simply be setup to run inside the container. A new version of the grading container was created building off of the previous grading container image. This meant that previous dependencies and software did not need to be reinstalled. A container was started up in interactive mode using the previous image version. Once running, the results CSV generation component file was copied into the running container. The component file was placed in the root directory of the container and the grading_dir directory previously created was left undisturbed. The pandas Python library was also installed inside the container because it is a

dependency of the CSV results component. After completing these tasks, the container was exited and used to create a new image.

Simplification using Command Line Scripts

Now, the new image of the container contains all of the necessary software to run the AutoGrader tool and execute the results CSV generation component. However, running all of these items required additional commands to be executed. When starting the container all of the directories must be mapped, the AutoGrader tool must be run, and the results CSV generation component must be run. All of this required information resulted in a lengthy command. Requiring users to type out such a long command each time they want to use the tool was unrealistic. In order to simplify the application launching process, a shell script was composed to handle all of this setup. Utilizing a shell script greatly simplifies the work required by users to launch the system. A shell script folder was added to the grading folder structure. Inside the main grading folder, a new folder named “startup” contains the shell script. This information is included in the zip file download made available to users of the system. Now, users simply execute the shell script from the command line to launch the container and all of its processes. A user can execute the command shown in Figure 6 from inside the main grading directory to begin the grading process.

```
./startup/run_grader.sh
```

Figure 6 - Startup Command. Type this command from inside the main grading directory to initiate the grading process.

It is easy to allow used Docker containers to pile up on the local machine hardrive. Overtime, this becomes a major storage issue. In order to prevent this, the '—rm' option was added to the docker run command contained in the startup script. This will automatically remove the container once it has exited, thus preventing the buildup of unused containers.

Testing of the system was initially performed in a Unix environment. Towards the end of the project, testing was conducted on a Windows system. The containerized grader is inherently portable, so this portion of the system had no issues running on Windows. Windows testing did identify two issues. First, the shell script used to launch the grader only works on Unix systems. The script was translated into a Windows batch script (.bat extension). This file replaced the shell script in the startup file. This also facilitated the need to have two separate zip files of the folder directory template; Windows version and Mac/Linux version. Both versions of the folder structure template are available from the GitHub repository. The startup command to call the Windows batch script is also slightly different due to the variations between operating systems. All of these differences are highlighted in the project documentation.

Project Documentation

Project documentation is a crucial component for effectively communicating information about the project. A project webpage is an excellent way to create a centralized location for project related information to live. Since this project has finished components located in multiple locations (Docker Hub and GitHub), the project webpage is important. A GitHub web page was created to provide documentation for this project. The web page discusses the contents of this

project and provides a user tutorial. The page begins with a description of the project and lists features of the tool. The installation process is clearly outlined and links to the Docker image and folder structure zip file are provided. A user tutorial is also provided. It walks through the entire process of setting up and running the containerized autograder. Snips are provided for multiple steps to make the tutorial easy to follow. Upon completion of this documentation page, several test users were tasked with completing the tutorial. This was done to verify that the instructions are clear and easy to follow. A snippet of the documentation page is shown in Figure 7.

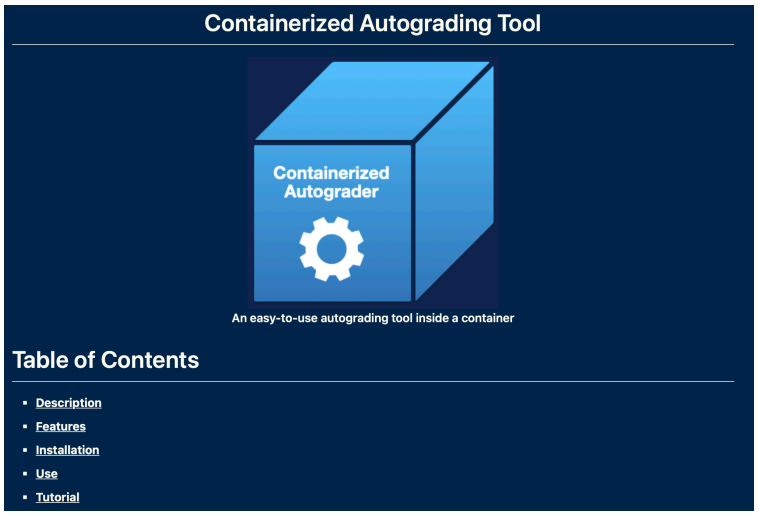


Figure 7 - Project Documentation Web Page

The documentation page also contains a common errors section. This lists several errors that occurred frequently during development and testing. The error name is provided along with an explanation of why the error occurred. The solution to rectify the error is also explained. This section was added to the documentation to further improve the usability of the system. A reference section is also provided that provides links to tools that were integrated into the final containerized autograder tool. This section was included to provide recognition to the tools that helped this project come to fruition. It also provides necessary information to individuals who

may wish to build a similar system using the tools from this project. This GitHub project page will help to share the knowledge gained from this project. It will also serve as a resource for people using the containerized autograder tool.

Student Use of the Containerized Autograder

The containerized autograder tool provides the resources necessary to permit students to self-check their assignments. Instructors can distribute a limited set of test cases for the students to use. The instructor may provide these files directly or add the test files to the grading directory structure prior to providing students with the complete directory for an assignment. Students can add their completed assignment to the grading directory and run the container on their assignment. Students can also use the same Docker image for the containerized autograder.

Allowing students to check their assignment prior to submission can be valuable. It gives students a chance to catch errors and gain confidence in their assignment. One disadvantage to permitting students the ability to use an autograder on their assignment is the possibility that students begin to rely on the autograder [1]. A student may attempt to reach solutions through brute force. Rather than thinking how to fix their assignment, they may try to make minor adjustments and resubmit to the autograder until they get a positive grade. The containerized autograder handles this situation by limiting the tests that the student has access to. Instructors can provide only a portion of the full test suite to students. This will ensure students continue to improve their submissions without abusing the system.

Chapter 4

Software Development Process & Project Management

An incremental development process was used for this project. Additionally, Agile methodologies were integrated into the project to complement the incremental development process. Under this process the specification, development, and validation activities occur concurrently. With the knowledge that this project must be completed in a short amount of time, a development process that allowed for feedback early and often was seen as beneficial. As the understanding of the project requirements evolved, this development process provided the necessary flexibility to adapt the project. Drawbacks of an incremental development process were also considered. The main issue with incremental development is the degradation of the software with each new increment. However, this issue is more critical for large projects [12]. Lastly, validation and testing procedures were put in place to mitigate this issue.

Agile methodologies were also integrated into the project development process. These methodologies allowed for more flexibility as project requirements changed. Also, Agile methodologies allow for early feedback which was vital to ensuring that the project was meeting its goals. A kanban board is a popular project management tool in Agile development. The board is a simple way to visually track the progress of tasks during the project. Traditionally, a kanban board is created on a white board where sticky notes are used to represent each task. There are three columns on the board; to-do, doing, and done. As tasks are created, the sticky note is placed in the to-do column. Once the task is assigned and work has begun on it, the sticky note is moved

into the doing column. Upon completion, the sticky note is placed in the done column. An electronic version of the kanban board was used during this project. The software is called Focalboard. A snapshot of the project kanban board is shown in Figure 7. This snapshot was taken approximately midway through the project. The tasks listed on this board are specific items that need to be completed rather than high level topics.

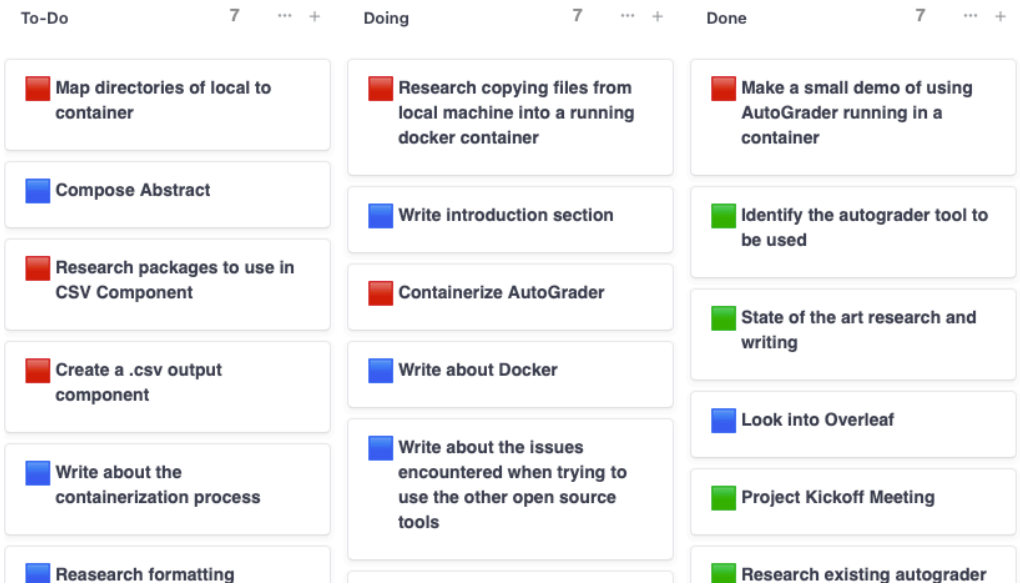
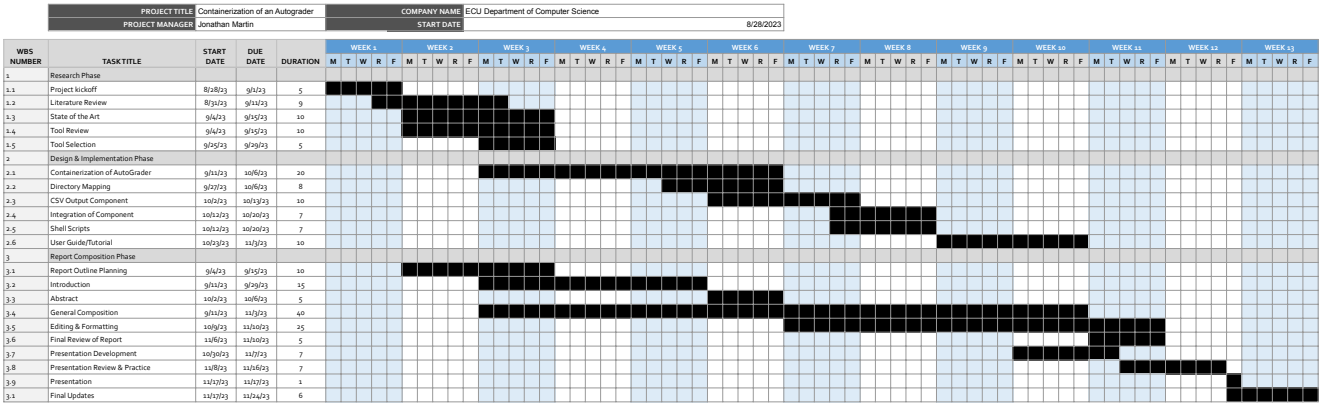


Figure 8 - Project Kanban Board

Agile development typically works in sprints. These are short time periods, typically 1 to 2 weeks, during which a handful of tasks are worked on to completion. This routine was employed during this project. New tasks were established during weekly meetings with the project advisor. During these meetings, tasks to be completed by the next meeting were determined and added to the to-do column of the kanban board. As work began on each task, it was moved to the doing column. As each task was completed during the week, it was moved into the done column. This schedule helped ensure that high priority tasks were completed quickly. It also helped track progress. Each task falls under a project phase. This board helped to ensure that all tasks that

belonged to a specific project phase were completed by the end date scheduled for each phase as shown on the Gantt chart for the project.

A comprehensive schedule was constructed at the onset of the project. A Gantt chart was used to create the schedule. A Gantt chart is a great way to visualize the project schedule as it shows the duration of each task over time. Figure 8 shows the Gantt chart for this project. The project was broken into three phases; Research, Design & Implementation, and Report Composition. Each of these phases contained several major tasks. These tasks are higher level than the tasks contained in the Kanban board. The research phase occurred mostly towards the beginning of the project while the other two phases lasted the remainder of the project. It is important to note that many of the tasks have overlapping durations. This symbolizes that these tasks were occurring concurrently. Task durations were estimated at the beginning of the project. However, the Gantt chart was updated regularly to reflect the actual progress made on each task. In general, the schedule was adhered to in order to meet the final deadline of the project.



PROJECT TITLE	Containerization of an Autograder	COMPANY NAME	ECU Department of Computer Science
PROJECT MANAGER	Jonathan Martin	START DATE	8/28/2023

WBS NUMBER	TASK TITLE	START DATE	DUE DATE	DURATION
1	Research Phase			
1.1	Project kickoff	8/28/23	9/1/23	5
1.2	Literature Review	8/31/23	9/11/23	9
1.3	State of the Art	9/4/23	9/15/23	10
1.4	Tool Review	9/4/23	9/15/23	10
1.5	Tool Selection	9/25/23	9/29/23	5
2	Design & Implementation Phase			
2.1	Containerization of AutoGrader	9/11/23	10/6/23	20
2.2	Directory Mapping	9/27/23	10/6/23	8
2.3	CSV Output Component	10/2/23	10/13/23	10
2.4	Integration of Component	10/12/23	10/20/23	7
2.5	Shell Scripts	10/12/23	10/20/23	7
2.6	User Guide/Tutorial	10/23/23	11/3/23	10
3	Report Composition Phase			
3.1	Report Outline Planning	9/4/23	9/15/23	10
3.2	Introduction	9/11/23	9/29/23	15
3.3	Abstract	10/2/23	10/6/23	5
3.4	General Composition	9/11/23	11/3/23	40
3.5	Editing & Formatting	10/9/23	11/10/23	25
3.6	Final Review of Report	11/6/23	11/10/23	5
3.7	Presentation Development	10/30/23	11/7/23	7
3.8	Presentation Review & Practice	11/8/23	11/16/23	7
3.9	Presentation	11/17/23	11/17/23	1
3.1	Final Updates	11/17/23	11/24/23	6

Figure 9 - Project Schedule

Chapter 5

Testing

Software testing is a vital part of the software development process. Testing was ongoing throughout project development. Testing early and often was the approach used for this project. Multiple levels of software testing were needed to validate various parts of the project. Unit testing was utilized to test components. Integration testing was needed for checking that the components worked together. System level testing was also necessary to ensure that the overall system met the needs of the project. Each of these testing types were deployed at various points of the project. User testing of the entire system was also conducted to ensure that the system meet the usability goals established for the project.

Unit testing was used during the development of the CSV results output component. This involved the testing of individual functions within the component. For example, the parse_names function shown in Figure 5 underwent unit testing. This required the creation of sample results files that were stored in a directory. This directory was then given to the function and the list returned by the function was checked against the actual names of the files. This process was repeated for each of the functions in the CSV results output component. Another specific test that was performed on the functions was verifying that each function could work with different file extensions. File extensions for all programming languages supported by AutoGrader were tested. Several additional file extensions were also tested to ensure that the tool can handle any future languages that may be added to the system. Conducting unit testing on each function as it was

created sped up the development of the component. Errors were caught immediately. Also, it was easier to fix the errors because the code was fresh in the mind of the developer. This isolated form of testing increased confidence that the component was functioning correctly, and future levels of testing could focus on the integration of the component.

Integration testing was extensive for this project. Integration testing is focused on checking the interactions between different components. This tool has several parts including a Docker image, shell script, AutoGrader application, and the CSV results output component. Integration testing was performed to ensure that all of these different parts of the system worked together correctly. Integration testing occurred every time two components were combined. The first wave of integration testing occurred when the AutoGrader tool was installed inside of the container. This testing was simply verifying that the AutoGrader tool still ran as expected inside of the Ubuntu container. Files were copied into the running container. Then AutoGrader was executed. The next integration test checked that the mapping of directories from local to container were working. The CSV results component integration aimed to confirm that the component was executed inside the container. It also verified that the component was able to find the results files inside the container. The shell script also underwent integration testing. This verified that all of the correct commands were fired upon execution of the script. This project involved a lot of components interacting with one another, so integration testing was crucial to ensure that everything was working together.

The iterative development process used for this project created multiple working versions of the system over the project lifespan. These were an excellent way to receive user feedback. Testing each version of the system helped identify additional improvements that were integrated into the project. One example of an improvement made due to system testing of an early version was the addition of the shell script. Each version of the system had a longer Docker run command than the previous version. After a few versions, it was apparent that this command was going to be cumbersome for a user to input. This led to research into methods for condensing Docker commands which resulted in the addition of the shell script for startup. Similarly, the '—rm', or, remove container option, was added after several weeks of development. It was noted that a large number of older containers had built up on the local machine and were taking up space. Instead of manually removing each old container, this tag was added. System testing was also helpful for gauging the progress of the project. The project schedule established deadlines for the completion of major development tasks. System tests were used to verify that these tasks had been completed.

System testing was conducted on both Mac and Windows systems. The container is able to run on any system the supports Docker, so compatibility with each operating system was a goal of the project. The AutoGrader tool provides several example grading directories. These contain sample assignments and tests for a variety of languages including C, C++, and Python. Each of these example grading directories was used to test the containerized autograder on each operating system. Mac testing did not encounter issues. However, Windows testing revealed several issues. First, the shell script was written in a language that Windows does not read. This

led to the translation of the script into Windows Batch script. Also, the startup command had to be modified to work on Windows. Windows systems use the backslash '\' symbol in the file paths whereas Unix based systems use the forward slash '/'. The startup command was updated to reflect this difference. Each operating system was testing from scratch by downloading the folder structure template for the respective operating system. The Docker image was retrieved from Docker Hub. Then testing of the tool proceeded. Testing was conducted in this manner to ensure that the correct resources could be retrieved from the references provided in the project documentation.

User testing was also conducted on the system. The main goal of this project was to make an autograder that is easy to use. Allowing actual users to attempt to use this system was an excellent way to check if this goal was met. The project documentation created for this project was also part of user testing. This documentation was used by users to install and use the system. Feedback was gathered for both the system and the project documentation. This information was used to improve the project documentation page.

Testing throughout the development phase allowed errors to be fixed as they were discovered. This prevented errors from propagating overtime and leading to technical debt. Testing early and often was also a form of risk mitigation. Errors are generally easier to fix early on during development. Due to the tight schedule of the project, it was important to reduce the chances of delays in development. Overall, testing helped ensure that a quality final product was created. User testing was also vital to ensuring that the project goals were met.

Chapter 6

Conclusion & Future Work

The result of this project is a containerized autograder. The image is publicly available on Docker Hub and ready for instructors to use. The goal of the project was to make an easy to use autograding tool. This goal was accomplished through the process of containerization. The frustrations associated with installing a tool have been reduced by utilizing Docker. Additional quality of life improvements were also added to further streamline the grading process.

A user of this system can quickly set up the system. Project documentation is provided on a GitHub webpage. This documentation includes installation instructions and a tutorial for grading. Links to the folder directory structure zip file and the Docker image are also provided on this page. The detailed explanations provided in the project documentation make it easy for users to follow. All of the required software is bundled inside the container, so users do not need to worry about installing dependencies. Users can focus on creating test cases and this tool will take care of the rest. The CSV results output file generated by the tool will also improve the grading process. This file contains the grades for all of the student submissions in a format that is compatible with Learning Management Systems. This will facilitate quick uploading of grades. After downloading the Docker image and the folder structure template, instructors can continue to use these items to grade multiple assignments. The installation process is only required once.

The containerized autograder creates an opportunity for students to self-assess their assignments prior to submission. An instructor can distribute a version of the grading folders that contain a limited set of the test cases for the assignment. Students will be able to run the container on their own assignment to see how well they have done. Based on this feedback, they can make improvements to their assignment prior to submission. Instructors will then use the full suite of test cases upon submission. This test suite will contain additional test cases that were not available to students.

The original AutoGrader tool has limited capacity on Windows operating systems. By running the tool inside a Docker container, this issue has been overcome. The containerized autograder can be used anywhere Docker is installed. This will make deployment of the tool easier for instructors and students.

All of the components developed during this project are open-source. This allows for future modifications by others. Instructors can adapt this tool and decide to make modifications for their specific situation. This may involve all or only certain components used to create the containerized autograder. This project also serves as a demonstration. It shows what the containerization of an autograding tool can look like.

There are many potential modifications to the underlying AutoGrader tool that can be made. Additional programming languages can be added to the tool. Developing automation techniques

for the generation of test cases from a solution file would also be beneficial. The CSV results output component can be modified to incorporate more output formats.

Deploying the containerized autograder to the cloud would also be beneficial. This setup could allow students to submit their assignments directly through an online portal. The assignments can be graded automatically and grades pushed to the learning management system in real time via the cloud.

The containerization of an autograder has yielded meaningful results. The tool created from this project is functional and available for use by instructors. The containerized autograder has minimal setup and execution requires a short command to be typed. The thorough project documentation will help onboard users and provide guidance to others hoping to build a similar tool. Creating a user friendly tool is vital to its success and the containerized autograder focused on achieving this goal.

Bibliography

[1] Baniassad, E., Zamprogno, L., Hall, B., & Holmes, R. (2021, March). Stop the (autograder) insanity: Regression penalties to deter autograder overreliance. In *Proceedings of the 52nd ACM technical symposium on computer science education* (pp. 1062-1068).

[2] CodeGrade - Deliver Engaging Feedback on Code. (n.d.). www.codegrade.com. https://www.codegrade.com/

[3] Codio | The Hands-On Platform for Computing & Tech Skills Education. (n.d.). www.codio.com. https://www.codio.com/

[4] codePost: Autograder and code review for computer science courses. (n.d.). Codepost.io. Retrieved November 2, 2023, from https://codepost.io/

[5] Edwards, S. H., & Perez-Quinones, M. A. (2008, June). Web-CAT: automatically grading programming assignments. In *Proceedings of the 13th annual conference on Innovation and technology in computer science education* (pp. 328-328).

[6] IBM. (n.d.). *Containerization Explained | IBM*. www.ibm.com. https://www.ibm.com/topics/containerization

[7] Ala-Mutka, K. M. (2005). A survey of automated assessment approaches for programming assignments. *Computer science education*, *15*(2), 83-102.

[8] OK. (n.d.). Okpy.org. Retrieved November 2, 2023, from https://okpy.org/

[9] Otter-Grader Documentation — Otter-Grader documentation. (n.d.). Otter-Grader.readthedocs.io. Retrieved November 2, 2023, from https://otter-grader.readthedocs.io/en/latest/

[10] Rad, B. B., Bhatti, H. J., & Ahmadi, M. (2017). An introduction to docker and analysis of its performance. *International Journal of Computer Science and Network Security (IJCSNS)*, *17*(3), 228.

[11] Russell, B. (2015). Passive Benchmarking with docker LXC. *KVM & OpenStack*.

[12] Sommerville, I. (2016). Software Engineering, 10/E. Pearson Education.

[13] Web-CAT - Web-CAT. (n.d.). Web-Cat.org. https://web-cat.org/projects/Web-CAT/