Project Anacapa: A Minimalist Autograder Based On GitHub

Hunter Laux, B.S.

A Project Submitted to the Department of Computer Science
of the University of California, Santa Barbara
in Partial Fulfillment of the Requirements for the
Degree of Master of Science

December 7, 2015

ABSTRACT

Automatic grading systems for programming assignments, *autograders*, are a time saving resource and a force multiplier for instructors and TAs. Students appreciate the instant feedback autograders provide on their assignments. An autograder should be easy to deploy, easy to learn for both the professor and the student, maintainable, and flexible. It is also helpful if it aligns with common development practices such as test-driven development (TDD) and continuous integration (CI). Existing autograding systems, including the current submit.cs system in place at UCSB, have a number of deficiencies, including ease of use, support, and maintenance challenges. The Anacapa grader takes a minimalist approach to developing an autograder, providing only the most necessary autograding functions with a minimum of source code and application infrastructure. We accomplish this by leveraging GitHub.com and classroom.GitHub.com for managing projects, submissions, configuration and feedback.

In this paper we will discuss the limitations of existing autograder systems. We will then describe how Anacapa overcomes these limitations, providing a scalable, easy to use system with a minimal amount of code to be maintained. We will explain the system both from an instructor and student perspective, show how the system is aligned with current industry practices, and provide an overview of support and maintenance concerns. We will conclude by describing suggestions for future work.

Contents

1 Introduction

Autograders are automatic grading systems for programming assignments. Autograded programming assignments allow instructors and TAs to spend less time on tedious validation of correctness and more time on content of the code, style, and/or enable them to serve more students. Where in the past, students may have had to wait days, or even weeks for feedback on a programming assignment, with autograded assignments, students receive immediate feedback on their work.

Autograders have their drawbacks as well. Code may be incorrect but still pass tests (verification vs. validation). Manual grading may be more effective at catching certain kinds of bugs. Students may rely too much on the system as a crutch rather than doing their own testing. TAs may not give individualized feedback for assignments on code style. Code might include “brute force” approaches, copied/pasted code that should have been refactored, inefficient implementations, etc.

We acknowledge these drawbacks, but note that they are out of scope for this project, which focused only on how to address certain shortcomings of a particular autograder system, in use at UC Santa Barbara—namely the submit.cs system developed by Bryce Boe [Boe, 2014]. In particular, the submit.cs system was found to present several challenges in terms of being sustainable and supportable: the source code is very complex, has no test coverage, and there was no clear roadmap for a “development/qa/production” lifecycle. While submit.cs currently provides many benefit to UCSB CS faculty/TAs and students, it is not clear how long the current system can be maintained and supported effectively.

We considered several alternative approaches to addressing the shortcomings of submit.cs. First, we first considered the current code base of submit.cs to address the shortcomings noted above. We judged that it would take more effort to refactor the code then it would take to rewrite submit using the current architecture using a framework such as Ruby on Rails. We also looked at an existing open-source autograder known as Web-CAT [Edwards, 2003], but found that Web-CAT had a similar maintenance and support issues (we say more about this in Section 3.5).

Finally we chose to develop a system based on a new architecture. In reviewing the implementation details of submit.cs, we found that the current architecture contained, as a subset, a large number of features of git and GitHub that had been “reimplemented from scratch”, including by not limited to, the deduplication feature of the submit.cs File Store module. This inspired us to consider how we might leverage the GitHub.com API to replace the File Store. As we examined this, we found more and more features of the current submit.cs that could be replaced by leveraging git, GitHub.com, and an existing open-source project created by GitHub called “GitHub Classroom”.

An autograder should be easy to deploy, easy to learn for both the professor and the student, maintainable, and flexible. It is also helpful if it aligns with common development practices such as test-driven development (TDD) and continuous integration (CI). Ideally, it should support any programming language that is available on the instructional lab systems used for the CS courses offered by the department. To take it a step further, ideally, any package, library, or other resource available on those systems should be automatically accessible in the context in which the autograder evaluates student submissions.

The Anacapa grader takes a minimalist approach to developing an autograder, providing only the most necessary autograding functions with a minimum of source code and application infrastructure. We accomplish this by leveraging GitHub.com and classroom.GitHub.com for managing projects, submissions, configuration and feedback. A feature that Anacapa grader shares with submit.cs—in fact, the only part of submit.cs that we reused—is the reliance on sandboxed ssh sessions on the existing UCSB CS instructional lab systems (CSIL). This features allows instructors the freedom to develop autograded assignments using any language, system, tool, or package available on CSIL.

Another desirable feature of a student grading system it that it utilizes development practices that students are likely to encountered in industry. Since git and GitHub are increasingly used for open and closed source development, GitHub is a natural choice on which to base a grading system. Further, the workflow of our grader system mirrors the industrial practice of Continuous Integration (CI), where code is checked against acceptance tests each time it is committed to the version control system.

Anacapa is built on Rails which is a popular development platform [Rails, 2015]. Additionally, Anacapa takes advantage of many Ruby libraries that significantly reduce the amount of code that needs to be maintained. Prof. Phill Conrad has a course scheduled for Winter 2016 (CMPTGCS 140, “Agile SAAS Development”) that will be based, in part, on training students in Rails development by having them participate in ongoing development, testing and maintenance of Anacapa Grader. The intent is for class to launch a self-sustaining development team managed by Dr. Conrad that can maintain this open-source project.

The rest of this project report is organized as follows. Section 2 provides an overview of existing grading systems, including their benefits and limitations. Section 3 describes the system from three perspectives: that of a instructor, that of a student, and that of a system administrator. Section 4 provides an overview of the Anacapa system architecture. Finally, Section 5 concludes the report with a summary of the major contributions of this project, some related work, and suggestions for future work.

2 Background on Autograder Systems

2.1 Evaluation of existing autograders

Many autograder systems exist or are under development. [Ihantola 2010] surveys seventeen (17) such systems, and sixty (60) papers published about autograders developed between 2006 and 2010. Section 5.2 (Related Work) provides a summary of their findings.

As a basis for a more detailed comparison, we chose to focus on two specific autograders:

- WebCat, developed at Virginia Tech [Edwards, 2003]
- submit.cs, developed at UCSB [Boe, 2014]

Although other grading systems exist, we chose to evaluate Web-CAT because it is the most mature and feature-rich grading system, providing many plugins. Additionally, we chose to evaluate submit.cs because it is a system currently deployed at UCSB for a number of courses and Anacapa's purpose is to be a candidate for replacement of the submit.cs system. For purposes of our evaluation, we do not consider commercial systems or closed-source systems.

2.1.1 Diff-Based grading and TDD frameworks

Both Submit.cs and Anacapa use diff-based grading. For a particular test a student receives points if there is no difference between expected output and the student's output.

Although Web-CAT provides a diff-based grading plugin, Web-CAT is more typically used with TDD frameworks such as JUnit and CxxUnit. Web-CAT grading reports can provide students with grades based on code coverage. Web-CAT also allow students to easily write their own tests cases. Grading according to these additional criteria provides an advantage over diff-based systems

Web-CAT's TDD framework based grading requires that a grading plugin for each TDD framework. Diff-based systems are significantly more flexible because any program that provides an output can be graded. For example, if a instructor prefers to use the GTest framework as opposed to CxxUnit framework for a C++ assignment significant effort would be required to write and install a new gtest Web-CAT plugin.

Since Anacapa is designed to be minimalistic a simple diff-based grading criteria is used. Testing frameworks can be used with a diff-based system, but output is statically compared. Grades are essentially pass/fail. Student supplied test cases are not considered for grading.

2.1.2 Student interface differences

Anacapa differs from other grading systems in that students use GitHub Classroom to submit assignments, which gives students relevant experience with git repositories as well as gives them the ability to use their favorite tools that integrate with GitHub. Since many development environments already integrate with GitHub students are provided with flexibility in the tools that she/he might want to use. For example, it is possible to use GitHub with source code collaboration services like c9.io. Since grades are posted to GitHub repositories students even can do everything within GitHub, including editing the code directly from the GitHub page. Basing Anacapa on GitHub maximizes the potential for flexibility and extendability.

Web-CAT has two main pieces: a web page and eclipse plugin. It does not require eclipse, but by using eclipse students can test code before submission and submit directly from an Eclipse plugin. Grades can be checked by viewing a web-page based report.

Similarly to Web-CAT, submit.cs provides a web page that allows students to check their grade. Submissions can either be generated from a web page based submission or a script that is installed on CSIL, allowing students to make submissions from a command line.

2.1.3 Instructor interface differences

Both submit.cs and Web-CAT require instructors to create projects via a web page. Anacapa allows an instructor to create projects in a GitHub repository. Using a git repository allows instructors to track changes to their assignments as they evolve over time. Additionally, it takes very little effort to push new assignments to the Anacapa system. In contrast both submit.cs and Web-CAT requires quite a few steps to publish new assignments.

Our decision to use a hand edited config file versus a web interface for assignment creation was both pragmatic and strategic. Simply put, we believe that computer science instructors prefer config files and git over web interfaces. Anecdotal evidence suggest that designing an assignment using this methodology takes significantly less time than setting up an assignment using the web-page based UI of a system such as submit.cs. Comparing these user interface methods could be a subject of future work. If it is found that a web interface is preferred, developing such an interface for Anacapa would be a straightforward task, though it would present more code to maintain.

2.2 Minimum essential functions for an autograder

In Agile development, there is a notion of “minimum viable product” (MVP). Our approach to developing an autograder was to identify and focus on this MVP.

In our view, the minimum viable product for an autograder is that it must be able to provide the following functions:

- As an instructor I can create a project definition, including instructor supplied files, student supplied files and grading criteria.
- As an instructor I can create a grade report for all students.
- As a student I can register an account that links to a verified school account.
- As a student I create submissions.
- As a student I can view autograding results.

2.3 Architectural Comparison of submit.cs with Anacapa Grader

Although we defer a detailed discussion of the architecture of Anacapa Grader to Section 3, for purposes of comparison, we provide a brief discussion now. Since Anacapa Grader relies on GitHub.com and GitHub Classroom for many essential functions, there are only two major system components:

- A Ruby on Rails application deployed on Heroku.com that provides a minimal user interface, and responds to push webhooks from GitHub.com. Since all of the data is stored in GitHub repositories all student grade reports are stored as GitHub markdown files(Readme.md). Additionally the project definition and the student submissions are also stored in GitHub repositories. Thus, the database is mostly relegated to storing mappings to GitHub credentials and the required web page views are minimal. Essentially, GitHub is providing most of the views for the system.

- A set of “workers” that were modeled after submit.cs system, which remotely executes code on CSIL machines for the purpose of isolation and maintaining a runtime environment. CSIL machines provide a running environment that matches the student's execution environment which is difficult to reproduce. Each worker runs under a different account on CSIL (i.e. submit2, submit3) for isolation purposes. Anacapa creates SSH sessions to CSIL machines that allows it to run untrusted student code. For testing purposes, any machine can act as a worker as long as it provides SSH key based authentication. In order facilitate the transition from submit.cs assignments to Anacapa and leverage the work done by Bryce Boe, several considerations were made to make the transition easier. This includes breaking up execution into a build phase and an execution phase as well as modeled the testable definitions after submit.cs.

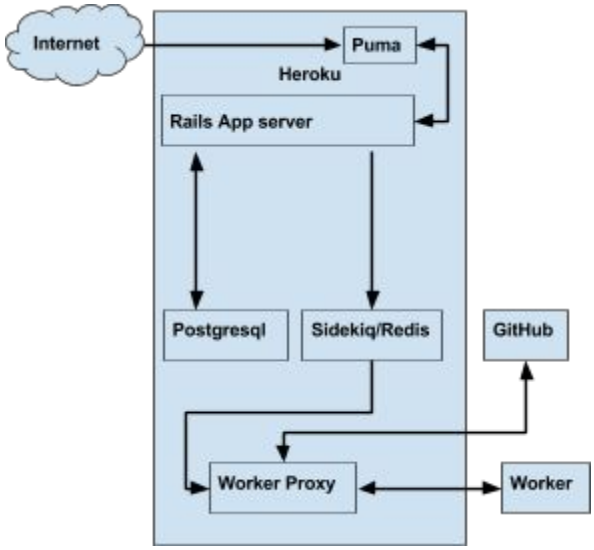
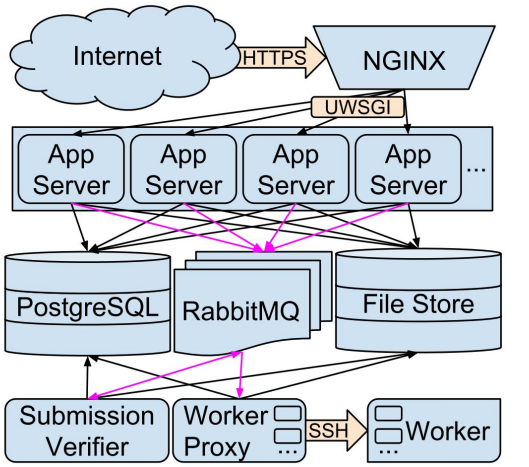
Anacapa Grader Architecture	submit.cs architecture (Figure 4.2 from [Boe, 2014])
	 **Figure 4.2:** Provides an overview of the system architecture and how components interact. Pink lines indicate messages being passed to and from the RabbitMQ service. Note that each *worker* runs in a separate isolated environment.

Figure 1: Architectural Comparison of submit.cs with Anacapa Grader

3 User perspectives on Anacapa Grader

3.1 Course Setup for Anacapa Grader

Since Anacapa leverages a significant amount of GitHub and GitHub Classroom, first we describe how these systems work. An instructor uses the GitHub Classroom web site to associate a GitHub organization with a particular course offering or “classroom”. Instructors can then set up each assignment in Github Classroom. For each assignment, Github Classroom provides the instructor a link to distribute to students. When each student clicks on that link, the student is directed to the GitHub.com login page, and upon logging in, a GitHub repository is automatically created in the correct organization for that particular assignment. Starter code is also automatically pushed to this repository. Students have write access to these repositories, but not administration access. Instructors have administrative access to the student repositories as administrators of the organization used for the classroom.

An instructor that wants to use Anacapa grader has to take one additional step beyond the “stock” setup for GitHub Classroom, which is to register the GitHub organization for the course with the Anacapa Grader website. This is one of only two human user facing user interfaces provided by Anacapa Grader.

The primary effect of registering a GitHub organization with Anacapa Grader is that Anacapa Grader sets up **GitHub webhooks** associated with the organization, and subsequently, every repository created in the organization. Because of the presence of these web hooks, each time a repository in the organization is created, or a commit it made to it, a message is sent to the Anacapa Grader backend web server. These webhooks make it possible for nearly instructors and students with Anacapa Grader to take place through the GitHub user interface, and through normal GitHub activities, chiefy committing code, rather than having to learn a separate user interface.

3.2 Repositories used by Anacapa Grader

There are five repository type used by Anacapa Grader, as shown in Table 1. The role of each of these repositories, the process by which they come into existence, and their relationships are presented in Figure 2, and explained in the remainder of this section.

The five types are repositories are:

1. **grader repository**: one per assignment. Created by instructor by hand editing. Contains reference implementation, and test case configuration. Available to instructors only.

2. **expected repository**: one per assignment. Automatically generated by Anacapa grader. Contains expected results for test cases. Available to instructors only.
3. **student repository**: one per student per assignment. Created by GItHub Classroom, then hand edited by student. Contains the student's actual work on the assignment.
4. **results repository:** one per student per assignment. Created automatically by Anacapa grader, and updated each time the student commits code to the student repository. Contains the actual results for each test case. Not directly accessible to students, since some test case output may be configured to be hidden from students. Full access for instructors.
5. **grade repository:** one per student per assignment. Created automatically by Anacapa Grader. Contains a formatted grade report for the student. Updated each time a student commits code to the student repository. Read only for student, full access for instructor.

Table 1: Anacapa Grader Repository Types

Role	Naming Convention	Instructor Access	Student Access	Created
Grader	grader-$(LABNAME)	Read/Write	None	manually, by instructor
Expected	expected-$(LABNAME)	Read/Write	None	Anacapa
Student	$(LABNAME)-$(STUDENT)	Read/Write	Read/Write	GitHub Classroom
Results	results-$(LABNAME)-$(STUDENT)	Read/Write	None	Anacapa
Grade	grade-$(LABNAME)-$(STUDENT)	Read/Write	Read Only	Anacapa

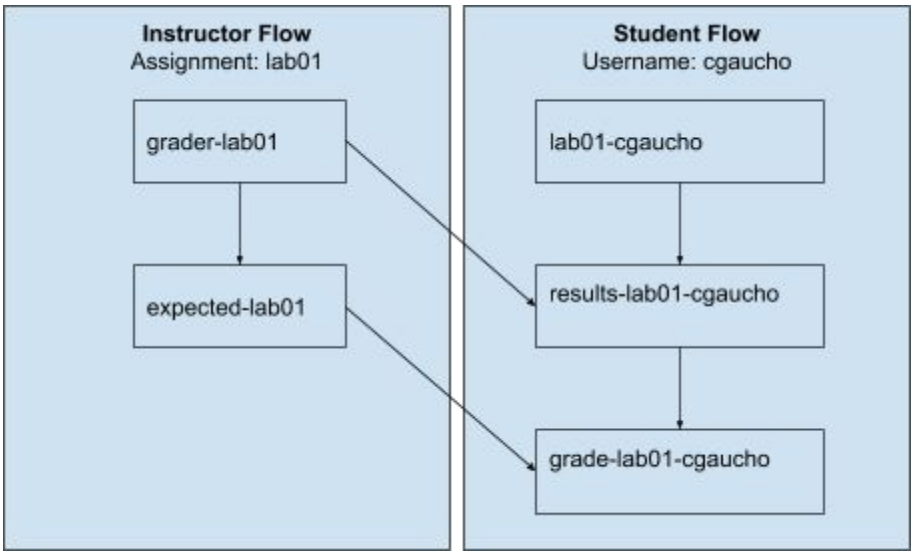


Figure 2: Repository workflow

Here is a more detailed explanation of each of the five repositories.

The instructor repositories must be generated before the student can begin working on their assignment. These are the **grader repositor**y, which the instructor creates and the **expected repository** that is automatically generated by Anacapa. The instructor manually creates the grader repository and places into it three things:

- A reference implementation of the work the student is expected to submit. This is used to generate the expected output.

- Files that are part of the submission that the student does not need to change. This can also include files that the student will not have access to such as data used in tests and/or test cases that are not provided to the student.

- A configuration file for the test definitions.

When the instructor creates and pushes any grader repository that matches the proper naming convention and contains the necessary configuration files, a message is automatically sent to the Anacapa grader. This message is generated by a “post-commit hook” that is associated with each repository in the organization. In response, Anacapa generates an expected repository based on the grader repository. It contains the automatically generated expected results which will be used for comparison when students submit assignments.

For a given assignment, there are three repositories that are associated with each student. The **student repository** is the repository automatically created by GitHub Classroom and is the one in which the student actually completes the assignment. The **results repository** contains the automatically generated output of the student’s assignment and is not visible to the student. The **grade repository** contains the final grade report for each student, which the student can view.

3.3 Instructor view of Anacapa Grader

The instructor must first add the classroom to the organization list in Anacapa. This is a one time step. The name of the organization added must match the one used in GitHub Classroom.

This can be done by visiting:
https://anacapa-grader.herokuapp.com/organizations/new

Next, the instructor must create a project grader repository. A sample can be obtained here:
https://GitHub.com/project-anacapa/grader-SAMPLE

The grader repository contains instructor files, student files and a testable.json file. The student files is a reference implementation of a student submission. The instructor files are files that are provided by the instructor which aren't intended to be modified by the student and the student will not have access to these files. The testable.json contains all the test definitions and grading criteria for the assignment.

After a grader repository has been created for an assignment the instructor can create a GitHub repository in the classroom organization using the naming convention grader-$(LABNAME) and push the test definition to this repository. After the push, an expected-$(LABNAME) repository will be auto generated in the organization.

The last step is to setup an assignment in GitHub Classroom by visiting https://classroom.GitHub.com/classrooms and creating the assignment using the labname and organizations from the previous steps. After this step the assignment is ready to go.

Grades can be checked by visiting the url described by this format: https://anacapa-grader.herokuapp.com/organizations/$(organization_id)/grade_reports/$(lab_name)

3.4 Student view of Anacapa Grader

A one time requirement for students is to visit the website where they register for the course, namely: http://anacapa-grader.herokuapp.com/register/ . They also must ensure that their umail.ucsb.edu email address is associated with their GitHub.com account (note that GitHub permits multiple email addresses to be associated with an account.)

When the student visits this link, they are asked whether they will give Anacapa Grader permission to view email addresses associated with the account. Anacapa uses this permission to match GitHub.com accounts with student identities as made available to instructors on UCSB systems such as eGrades (the grade reporting system) and Gauchospace (Moodle), the campus LMS.

The instructor provides an assignment link when creating a GitHub Classroom assignment. The student will have a project repository after clicking on the classroom link. GitHub Classroom provides the student with write access to this repository write by making the student a collaborator. Shortly after submitting to the student repository a student will receive their grade in a automatically generated grade repository. Since students should not be allowed to change their grade, the student is added as a read-only collaborator to the grade repository as a Readme.md.

3.5 System Administrator view of Anacapa Grader

Anacapa is based on the Ruby on Rails framework, which provides significant advantages when it comes to maintenances and deployability over systems like submit.cs and Web-CAT.

	submit.cs	Web-CAT	Anacapa Grader
Language	Python	Java	Ruby
Framework	Pyramid	WebObjects	Rails
Source files	72	1103	54
Lines of code	7,245	282,135	1,600

Web-CAT written in Java is based on the WebObjects framework—a framework initialized released in 1996, and that has had no active development since 2008. The code based for Web-CAT is rather large, consisting of over 1103 Java and HTML files. Submit.cs is written in Python and is based on the Pyramid framework and is divided up into 72 Python and Python Template files consisting of 7,245 lines of code.

Anacapa on the other hand consists of 54 Ruby and Embedded Ruby (.erb) files and 1600 lines of code, not including test cases. (Submit.cs does not have any test cases.). We should acknowledge that Anacapa is not quite production ready at the time of this writing, and the code base is likely to grow.

In contrast with in Anacapa, submit.cs has several ad-hoc dependences including SQLAlchemy for the object relational mapping (ORM) and Alembic for migrations. Rails integrates both the ORM and migrations into a single framework, which many developers are already familiar with.

Rails apps are also significantly easier to deploy on cloud services like Heroku. Web-CAT being based on WebObjects requires significant effort to deploy in cloud based systems like Heroku. Virginia Tech allows other institutions to use their servers, which suggest that deploying and maintaining your own server can be difficult.

Deploying submit.cs on Heroku is possible, but challenging. The submit.cs system depends on the python-ldap package, and that package depends on native libraries. A build package is required just to compile submit.cs for Heroku. The submit.cs system also requires

non-ephemeral filesystem storage, so files are automatically lost when a Heroku dyno is restarted.

Anacapa deployment on Heroku by contrast is straightforward, since it was designed to run on cloud-based infrastructure from its conception.

Backing up Anacapa's database is also trivial since there are already a number of ways to backup Postgresql databases on Heroku, and it is relatively small in size—and contains very little mission critical data. It actually provides only a mapping from GitHub user accounts to access tokens used for authentication in the GitHub API. Further, even in the worst case scenario of total data loss, the only thing necessary to completely recover would be for the instructor to register each of her/his courses once, and for each user to register his/her GitHub account once. Those two actions would be all that would be necessary to completely restore the system. This is because the most artifacts of the Anacapa system are stored in GitHub.com repositories. Note that there isn't really any need to reestablish credentials from previous quarters, since there are no active users, and all artifacts of student work and student grades are maintained in GitHub repositories.

In order to backup submit.cs, not only would does the postgresql database need to be backed up, but the filestore must be backed up as well. This file store contains a copy of every file ever submitted to submit.cs and although deduplication techniques are used to minimize the size, no compression techniques are used to minimize the size of this file store. It's growing and shows no signs of stopping. Anacapa uses GitHub for the purposes of storing student submissions and reports, so the system administrator is not burdened with the task of backing up a file store or providing more storage when the files store grows beyond the capacity of the drive.

4 Architecture of Anacapa Grader

GitHub Classroom allows students to submit assignments fairly easily, but it is lacking a autograder system. Anacapa is that autograder system. By leveraging features present in GitHub, we were able to keep the implementation of Anacapa fairly minimal.

In order to leverage a GitHub Classroom, Anacapa hooks into the same organization that is used in by Classroom via an organization-level webhook. When a GitHub webhook event is triggered, GitHub does an http request to the Anacapa web server. The other feature that Anacapa takes advantage of is the GitHub API which allows Anacapa to administer repositories, read user information and add webhooks.

A student submits an assignment by pushing their git repository containing their code to the student's classroom repository. Since the repository exists in the organization, any push event to a student's repository triggers the organizational webhook. We also store instructor project definitions in repositories in the same organization (though with security settings that prevent student access) and use the same webhook to process instructor grader repositories.

An instructor will create a grader repository, which is processed into an expected results repository. Anacapa is aware of the naming convention GitHub Classroom uses to name student repositories and also adds naming conventions of it's own for other repositories used for calculation of student grades. Thus, Anacapa can process a repository differently depending on what kind of repository is being committed to.

The instructor flow starts with a grader repository that the instructor creates manually. Pushes to this repository generate webhook events that, so that Anacapa can automatically create an expected results repository. The student flow starts with a assignment repository that is generated from GitHub Classroom. A push to this repository generates a push webhook event which triggers Anacapa to push to a results repository. Subsequently the results repository generates another webhook event which triggers Anacapa to build a grade repository with the student results. The purpose of the intermediate results repository is to hold the output artifact from all the test runs, since the instructor may elect to hide these results from the student.

4.1 Storing GitHub Credentials

The primary purpose of the database on Anacapa is to map organization to GitHub access tokens. The GitHub OAuth web flow allows Anacapa to get an OAuth access token which can be used to access GitHub API on the behalf of a particular user. Additionally, the database stores mappings from organizations names to organization instructors. The instructor must have administrative access to the GitHub repository. By using the instructor GitHub access token Anacapa is able to perform the required grading tasks on behalf of the instructor.

4.2. GitHub Permissions

Anacapa uses the GitHub API to get access to GitHub resources that an instructor or student has access to. When an instructor or student logs in, an access token is generated that grants Anacapa permissions to read and write to organizational repositories, and install webhooks.

In order to access certain parts of the GitHub API applications can request certain *scopes,* i.e. defined subsets of information access. Unfortunately many of these scopes are very broad, but it is currently the only way that GitHub allows applications to access the organizational repositories that are used by GitHub Classroom. Current required scopes for Anacapa are:

- repo - For reading/writing private organizational repositories that the instructor has access to.
- admin:org - listing all organizations
- admin:org_hook - install organizational web-hook

The student scopes required for retrieving student e-mail addresses are:

- user:email - Finding all email addresses associated with a user account.

4.3. Mapping users to grades

One problem with using GitHub is there must be a mechanism for linking student accounts to grades. The mechanism submit.cs uses is the UCSB ldap server.. The primary purpose for this mapping is, so that grade reports can be generated for each student and the grade can be associated with a student identity in other grading systems such as GauchoSpace.

In place of this, Anacapa uses a minimalist approach that has been commonly used in many systems. GitHub allows for multiple email addresses associated with an account and uses a simple email verification method to ensure the student email account belongs to the student. Note that since Anacapa is hosted on Heroku, which is an off-campus server, UCSB ldap access is not an option without clearing significant administrative hurdles.

The database additionally stores mappings from students to access tokens that are necessary for retrieving emails used for determining student identity. UCSB provides each student with a user account and an associated email account(i.e. <user>@umail.ucsb.edu). Unfortunately GitHub does not provide a separate API to verify school enrollment, although this same technique is used for their educational discount program. The simple method Anacapa uses to establish identity is to search the entire email list for each student for the school email address associated with a GitHub ID. Grade Reports can then be generated automatically with UCSB net IDs by querying the GitHub API for mappings between student accounts and searching for a list of grade repositories in the GitHub organization.

5 Conclusion

5.1 Summary of major contributions

In this report, we compared Anacapa to other autograders, described how users interact with Anacapa, and described the mechanisms Anacapa uses to integrates with GitHub and GitHub Classroom. Since user interactions with Anacapa is utilizes GitHub APIs and GitHub Classroom it is able to achieve the minimal viable product with relatively few lines of code, leveraging tools that are already supported by GitHub and the open-source community. Anacapa provides much of the same functionality as Web-CAT and submit.cs does, but is open to extendability without significant support cost in an agile fashion. Anacapa provides the functionality that is missing in GitHub Classroom, which is the autograder and makes it a much more useful product which will help make students get engaged and save time for professors.

5.2 Related work

Earlier sections of this paper already cited the submit.cs system [Boe, 2014] and Web-CAT [Edwards, 2003] and important related work on autograders. In fact, autograders are a subject of a great deal of interest. As Armando Fox et al. wrote in a paper that appeared in a recent workshop [Fox, et al. 2015]

> "Given that 17 autograding systems and over 60 papers about them were produced from 2006–2010 alone [11], why did we choose to build our own?"

The paper cited as [11] in this quotation is [Ihantola, 2010] which does indeed provide a review of autograder systems developed between 2006–2010. From this paper we learned that *output comparison* (the technique we call "diff-based grading) is the most typical form of autograding. We learned that output comparison systems are typically language independent. The paper also described other forms of testing beyond output comparison and XUnit tests such as the JUnit and CxxUnit tests in Web-CAT. The paper describes various techniques used by grading systems to prevent mindless trial and error submission like those observed in [Boe, 2014] with students using the submit.cs system. The authors of [Ihantola, 2010] were also disappointed by the closed-source nature of most autograders, so we are glad that Anacapa does not contribute to this problem.

Meanwhile, [Fox, et al. 2015] goes on to explain the reasons they developed their own system, several of which apply equally to our effort. The authors explain that many existing systems are tightly integrated with a Learning Management System (LMS). (The local UCSB LMS is Gauchospace, based on Moodle.) They express a concern that the autograder should be instead insulated from the LMS for security reasons, i.e. to avoid the risk of untrusted student code compromising the integrity of the LMS. They also expressed the concern that while their

autograder needed to be insulated from LMS, it still needed to integrate with it in order to communicate results to students. We have the same needs, but our needs are particular to the systems used at UCSB both for identifying students (e.g. perm number, umail address) and for managing courses (eGrades, Gauchospace.)

A difference between the needs expressed by [Fox, et al. 2015] and our needs is that they required an autograder that would work at the scale of a Massive Open Online Course (MOOC). Our system has a much smaller scale requirement. The limiting factor of our system is the workers which are not currently deployed in the cloud. However, given that our implementation is based on a rails app that can run on cloud infrastructure (Heroku) and utilizes cloud services (GitHub), and that the number of worker processes is not a priori limited in any way, we see no particular obstacles to scaling our architecture.

Among the drawbacks we cited of autograder systems is that they do not address issues of coding style. An approach to coding style is represented in [Moghadam et al., 2015], which describes how an instructor can use a user interface to train a perceptron on how to grade programming style for MOOC style courses.

5.3 Suggestions for future work

An important consideration that has not yet been explored is pair and group assignments. This is a feature present in submit.cs that is perhaps the first important enhancement beyond “minimum viable product”. There is a feature for group assignments in GitHub Classroom, though we have not yet explored how it works, or how it might be integrated with Anacapa Grader. We leave that as future work, with the observation that is is reasonably likely that it may be rather straightforward.

For those who wish to use a UI to generate assignments, we suggest a web based UI which constructs grader repositories, similar to the instructors UI in submit.cs. This may increase adoption since many people still like to use UIs and it would help teach instructors how to use the system. Providing such a UI does not preclude allowing instructors that wish to continue hand editing the configuration to continue to do so, and would be a straightforward enhancement. Ideally, it could be developed in a way that it as loosely coupled to the rest of Anacapa as possible.

A benefit of being extremely light weight is that Anacapa can be easily extended. Our last suggestion for future work is to provide a plugin framework similar to Web-CAT, but easier to use. The primary modification to the system would be the way that the grade report gets generated in the last step by comparing the differences of the two functions. Since Web-CAT requires an understanding of the intricacies of the system to build new plugins as well as system administrative access to install. We propose a plugin framework that an instructor can administer and one that is easy to use in the context of the current system.

Bibliography

[Boe, 2014] Boe, Bryce A. "Enabling Wide-Scale Computer Science Education through Improved Automated Assessment Tools." Order No. 3645611 University of California, Santa Barbara, 2014. Ann Arbor: ProQuest. Web. 4 Dec. 2015.

[Edwards, 2003] S. H. Edwards. Rethinking computer science education from a test-first perspective. In Companion of the 18th annual ACM SIGPLAN conference on Object oriented programming, systems, languages, and applications, OOPSLA '03, pages 148–155. ACM, 2003.

[Fox, et al. 2015] A. Fox, D. A. Patterson, S. Joseph, and P. McCulloch, "MAGIC: Massive Automated Grading in the Cloud," in CHANGEE (Facing the challenges of assessing 21st century skills in the newly emerging educational ecosystem) workshop at {EC-TEL} 2015, 2015.

[GitHub, 2015] GitHub API v3 | GitHub Developer Guide. Retrieved December 4, 2015, from https://developer.GitHub.com/v3/

[Heroku, 2015] Getting Started with Rails 4.x on Heroku | Heroku Dev Center. Retrieved December 4, 2015, from https://devcenter.heroku.com/articles/getting-started-with-rails4

[Ihantola 2010] Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä. 2010. Review of recent systems for automatic assessment of programming assignments. In Proceedings of the 10th Koli Calling International Conference on Computing Education Research (Koli Calling '10). ACM, New York, NY, USA, 86-93. DOI=http://dx.doi.org/10.1145/1930464.1930480

[Moghadam, et al., 2015] Joseph Bahman Moghadam, Rohan Roy Choudhury, HeZheng Yin, and Armando Fox. 2015. AutoStyle: Toward Coding Style Feedback at Scale. In Proceedings of the Second (2015) ACM Conference on Learning @ Scale (L@S '15). ACM, New York, NY, USA, 261-266. DOI=http://dx.doi.org/10.1145/2724660.2728672

[Rails, 2015] Ruby on Rails Guides. Retrieved December 4, 2015, from http://guides.rubyonrails.org