Project Plan: Version 2

December 15-21

COGS (Central Online Grading System)

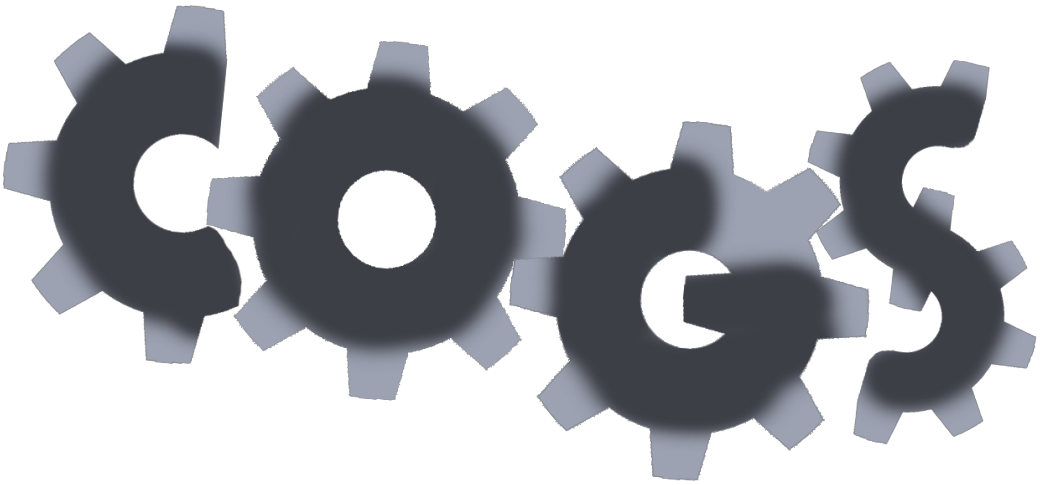

Table of Contents:

1. Team Overview

Kathryn Widen - Computer Engineering
kwiden@iastate.edu

Daniel Riechers - Software Engineering
riechers@iastate.edu

Daniel McDonough - Computer Engineering
dmac@iastate.edu

Forrest Scott - Computer Engineering
fvscott@iastate.edu

Zachary Lones – Computer Engineering
lones@iastate.edu

2. Problem Statement

The main goal of this project is to create a web application that will drastically reduce the amount of time required for Professor Daniels and his TAs to grade code submitted by students in CPR E 185. Professor Daniels would like for students to be able to come up with their own ideas rather than assigning homework that will give specific outputs given specific inputs, it is therefore highly unlikely that the webapp will be an autograder in the sense that it will grade student's code.

We wish to create an environment where TAs can be assigned groups of students and view the work from each student and grade them as well while minimizing the number of actions needed by the TA to complete the task. The grades submitted by the TA will then be sent to Blackboard for record keeping and so students can view their grade and any comments.

The webapp will adhere to all legalities related to safety of students' grades.

3. Definitions

Server: Hosts grader, web interface, and score database
Grader: Compiles and runs student code, is not a human
Report: The output of the compiler and code as well as source code. (ie compile errors, and code output) This is what the instructor uses to score submissions.
Student: Person who submits code to COGS for review
Instructor: Person who uses grader report to assign a score to submissions.
Head Instructor(s): Person with slightly more power, allowed to submit grades to Bb
Submission: A student's single attempt at an assignment.
Unit Tests: Automated tests performed by the grader on student code (not to be confused with unit testing for testing the functionality of the system.)
COTS: Commercial off the shelf.
RPM: Redhat ® Package Manager. RPM files are used to package and deliver compiled software.
SRPM: Source Redhat ® Package Manager. SRPM files are used to package and deliver source code. SRPMs include source code and build configuration data.
ISO: An image file used to record data onto compact discs.
Kickstart: A service for automating the installation and configuration of Redhat ® based systems.

4. System Description

COGS consists of several components:

- Back End: The backend system will be managing all of the data between instructors, students, and black board. The system will be capable of:
    - Interfacing with Blackboard to update student grades
    - Use Pubcookie to verify users upon login and classifying the users as admin, instructor/TA, or student.
    - Upload and store student submissions
    - Compile and auto-grade submissions
    - Store or immediately upload student grades to blackboard
- Front End: The system will be accessible through a webpage hosted on ISU servers. The front end will have three distinct parts:
    - Student Front End - Any user that is identified as a student will use COGS to upload assignments.
        i) Submit/Resubmit assignments
        ii) View feedback from auto-grader
    - Instructor Front End - The instructor front end will be viewable by those with instructor privileges. The instructor will be the heaviest users of COGS, and so the system should be set up to make their workflow the smoothest
        i) Add/remove/edit assignments
            (1) add/change/remove assignment due date
            (2) mark assignment to be auto-graded or instructor graded

(3) set criteria/rubric for assignments

ii) view/grade new submissions that they are authorized to see

iii) view/grade old submissions that they are authorized to see

iv) View assignment submitter name and class section

- Admin Front End - The admin front end will have all the same functionality as an instructor users with added privileges:
  i) add/remove instructor or student users
  ii) add/remove admin users
  iii) manage individual instructor privileges

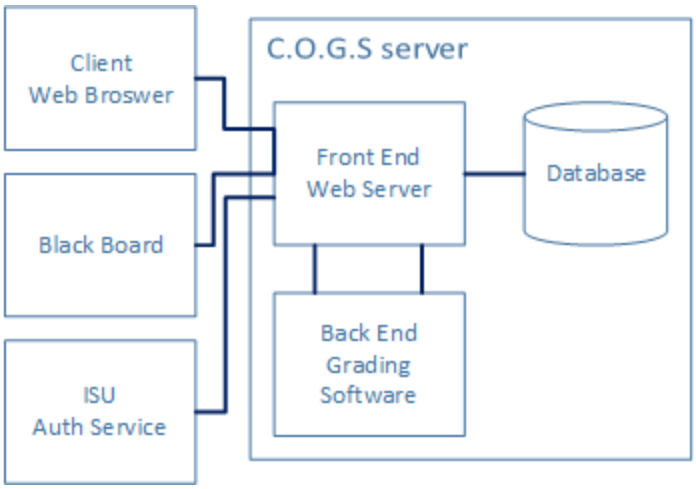

Figure 4.1 - System Overview Diagram

5. System Breakdown

Deliverables

System Deliverables:

- Iso containing all necessary COTS packages and dependencies
  - Install from iso automated with Kickstart
  - All system configuration automated with Kickstart
- COGS System:
  - Secure encapsulated environment for running student code
  - Compiler
    i. Support for gcc
    ii. Support for clang
  - Mandatory Access Control Policy
    i. Transparent to administrators
    ii. Tightly secured environment for running encapsulated code.

Software Deliverables:

- Software delivered in RPMs
    - RPM ensures correct permissions and configuration
    - SRPM keeps all build configuration in one place so it is easily repeatable
- Software should be added to system iso
    - Added to iso's yum repository as a
- COGS Software:
    - Analysis and scoring program
    - Web UI
    - Blackboard integration software

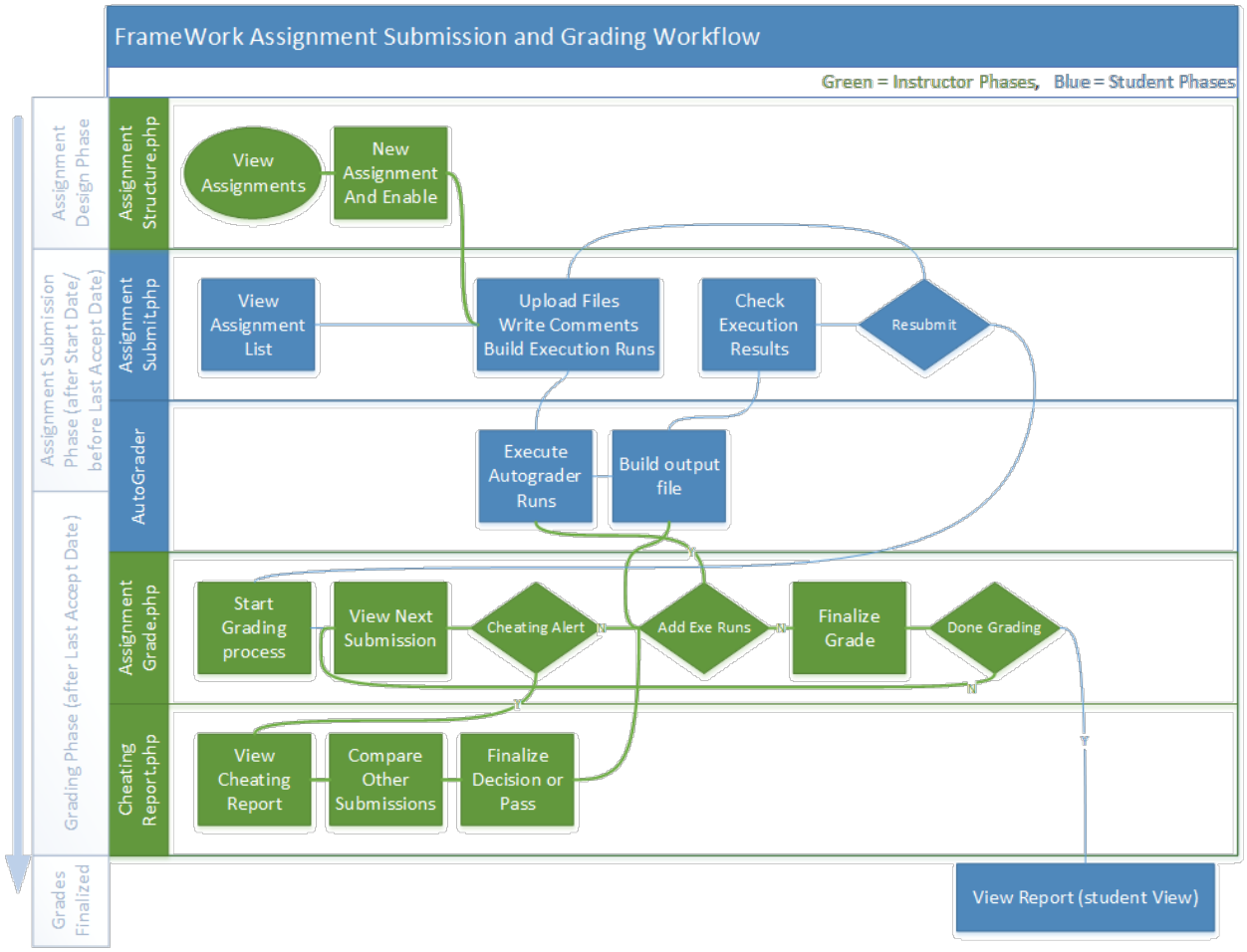

Figure 5.1 - System Workflow Diagram

6. Specifications

User Specifications:

- Students must be able to access the system to submit code for current assignments.

- Students should not be able to submit code for previous assignments past due dates
- Professors and TA's must be able to easily access computed scores and code of students.
- Professors TA's must be able to manually change given scores
- Professors must be able add and modify due dates and assignments

System Specifications

- Cogs must store scores as securely as possible
- Cogs must make the best effort to not be exploited by students' code
- Cogs must automatically analyze submitted code, generate a score and provide any relevant feedback to students.

7. Schedule and Gantt Chart

PHP Framework

- **ORM**: February - April
- **Controllers**: March - June
- **Blackboard API**: May - September
- **Accounts/Security**: August - November

Grader

- **Cheating Detector**: - April - May
- **Scorer/Unit Tests**: April - May
- **Command line interface**: April - May

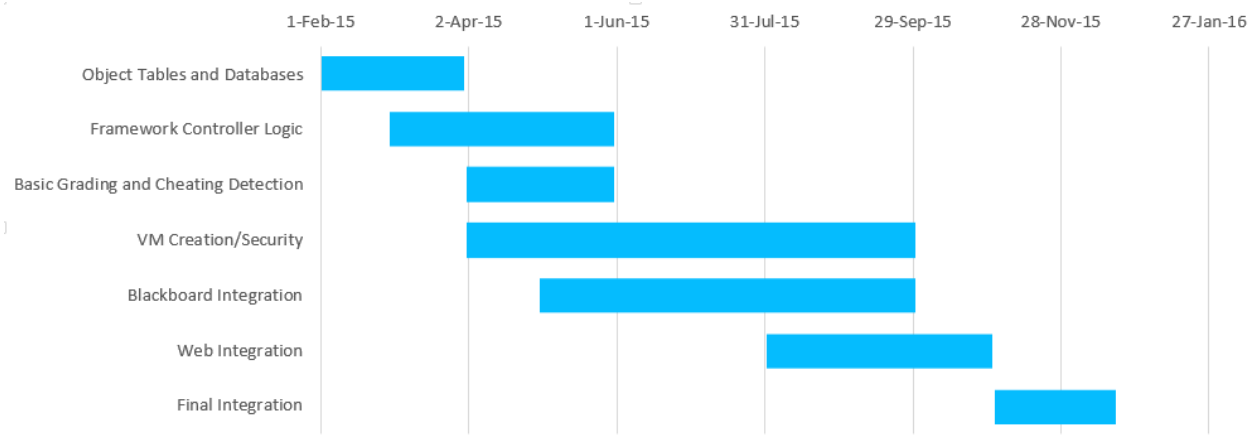

Figure 7.1 - Gantt Chart

8. Resources Required

The system will require a virtual server hosted by ISU. This would allow the system to be scalable across multiple classes and sections.

Testing

The system will need to undergo semi-rigorous testing before being pushed into production. For development purposes, the COGS team has access to a test class full of “dummy” students to test system integration and individual components.

The best way to test the full system is to do a “soft opening”. Like when a restaurant does a soft opening, the system will go live and can be used, but more for testing than for function. This soft opening will allow for a group of students and instructors to use the system under low stress/low load situations and will allow for the whole spectrum of tasks supported by the system to be tested. We will request feedback from all users to help us identify what needs to be changed before the system is pushed to production. This testing would be done across multiple instructors.

9. Risks

Security Risks

Since the data our server will be processing could contain personal data protected by law, it is of the utmost importance to implement secure designs. While our goals reflect realistic and obtainable security levels, there is always risk involved with possible intrusion attempts on our server.

Hosting Risks

There are also risks involved with bureaucracy and possibly unknown rules related to hosting. The university may demand that the server is hosted within a virtual machine stack and tied to protected databases. It will of course be possible to accommodate any hosting requirements, but there are risks involved with establishing a long-term hosting solution

BlackBoard Risks

The designs suggest some level of integration with the BlackBoard services. There are risks and involved with obtaining permissions to interface with the BlackBoard system, as well as unknowns about the availability of features and usability with the BlackBoard API. Also, any time our system would write to the BlackBoard system, it is essential that great care is taken in order to prevent any false records or wrongful deletion of current records within the BlackBoard system.

Feasibility Risks

The goals of this project are extensive, which means careful planning with have to take place in order for different modules to come together, as well as guaranteeing that the highest priority features are completed first. By implementing the project in phases, the project can prove the feasibility of each added feature while simultaneously providing clear separation within the design layers. Lastly, our plan must reflect the possibility that if not all of the stretch goals are reached, the core functionality will still be provided.

Technology Integration Risks

Our project spans many different technologies from information security tools, to compilers, to web frameworks, to possibly chrome addons. There will be risks created at any time there are many different technologies and APIs being bonded together within one endeavor. these risks can stem from version incompatibilities, messaging standards, or just simply from misunderstanding the time needed to work out the complexities of a new technology. Planning out how each of the many pieces will fit together is crucial to minimize the chances of a new technology subverting the entire project.