QuickFeed on Programming Assignments

Hein Meling

University of Stavanger, 4036 Stavanger, Norway hein.meling@uis.no

1 Introduction

In the last decade, programming has become an increasingly important tool for almost all science and engineering disciplines. To this end, programming exercises have become an essential tool for students to learn the craft of programming and apply, model, and evaluate other scientific techniques.

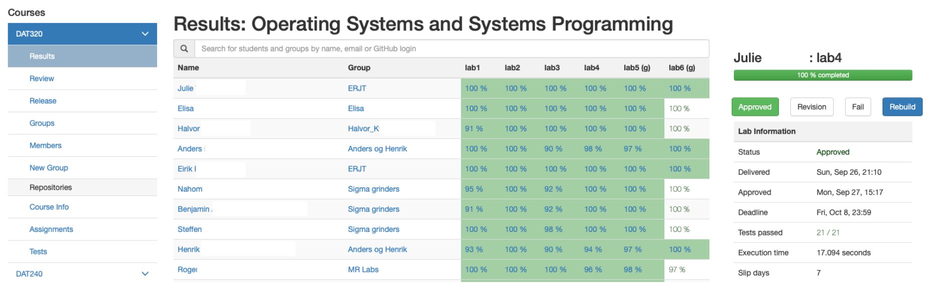
Unlike traditional assignments typically handed in on paper, programming assignments hold the unique property that their content and correctness can be evaluated automatically. This allows students to receive immediate feedback on their assignments. Instead of waiting days, or even weeks for assignments to be evaluated, students can receive feedback within minutes or even seconds. This allows students to submit assignments repeatedly and rapidly discover and fix their mistakes. Further, the feedback received from the automatic evaluation is not limited to a simple pass/fail decision. This allows students to work in a new manner, where instead of submitting an assignment once, they can repeatedly submit partial solutions and observe how their score improves. This helps students to find mistakes early and provides motivation.

The goal of QuickFeed is to strengthen the learning outcomes for students in programming courses with near-instantaneous feedback to students. The author has led a multi-year effort to develop and maintain QuickFeed, a web-based system for providing feedback on submitted code assignments. This poster aims to summarize QuickFeed's features and some of the technical solutions that others may find interesting.

2 QuickFeed's Approach

The University of Stavanger uses QuickFeed in six different courses. Figure 1 shows a screenshot of a small portion of QuickFeed's teacher view. In the following, we outline how students and teachers interact with QuickFeed.

How Students Interact With QuickFeed: To use QuickFeed, students must sign up for an account via GitHub. Once students have signed up, they can access programming assignments published by the teaching staff in a course. There are two types of assignments, individual and group assignments. Students must then try to solve the assignments within the deadline for each assignment. When students submit a solution to an assignment, this solution is picked up by the QuickFeed system, which runs a predefined set of solution checkers against the submitted code, in order to validate its correctness, and to give a weighted score



Results: Operating Systems and Systems Programming

Name	Group	lab1	lab2	lab3	lab4	lab5 (g)	lab6 (g)
Julie	ERJT	100 %	100 %	100 %	100 %	100 %	100 %
Elisa	Elisa	100 %	100 %	100 %	100 %	100 %	100 %
Halvor	Halvor_K	91 %	100 %	100 %	100 %	100 %	100 %
Anders	Anders og Henrik	100 %	100 %	90 %	98 %	97 %	100 %
Eirik	ERJT	100 %	100 %	100 %	100 %	100 %	100 %
Nahom	Sigma grinders	95 %	100 %	92 %	100 %	100 %	100 %
Benjamin	Sigma grinders	91 %	100 %	92 %	100 %	100 %	100 %
Steffen	Sigma grinders	100 %	100 %	98 %	100 %	100 %	100 %
Henrik	Anders og Henrik	93 %	100 %	90 %	94 %	97 %	100 %
Roger	MR Labs	100 %	100 %	100 %	96 %	98 %	97 %

Julie : lab4

100 % completed

Lab Information	
Status	Approved
Delivered	Sun, Sep 26, 21:10
Approved	Mon, Sep 27, 15:17
Deadline	Fri, Oct 8, 23:59
Tests passed	21 / 21
Execution time	17.094 seconds
Slip days	7

Fig. 1: QuickFeed screenshots.

for the different checks and a total percentage score for the whole assignment. Typically, a submission with a total score of 90 % is considered sufficient to pass a given assignment. QuickFeed will display a textual description explaining what the student's code is expected to do if a check fails. This feedback usually arrives in less than 30 seconds. Based on the feedback, students may revise their solution and resubmit multiple times until they are satisfied. In our experience, most students will continue to resubmit until they reach 100 % on the assignment.

How Teachers Interact With QuickFeed: The teaching staff of a course must do a fair bit of preparatory work before assignments can be published and provide automated feedback.

The teaching staff must create the course on GitHub and QuickFeed. When creating a course, QuickFeed creates separate repositories for *assignments* and *solution checkers*. The assignments repository will be made available to all course students and is populated with assignment descriptions and code templates that students may need to solve the assignments. The repository for solution checkers is only accessible to the teaching staff and the QuickFeed system. Such solution checkers must also be developed by the teaching staff and comprise a set of checks to evaluate and score student submissions. These checks must be designed so that they can provide textual responses to the students explaining why their solution failed, and at the same time, provide a weighted score for the checks that passed. Further, each assignment can be configured in various ways, e.g., individual vs. group, minimum passing score, automated vs. manual approval, and more.

During the course, teachers can view the results achieved by each student or group, and can manually approve assignments. Current practice has been to automate approval for the first few assignments, with the remaining assignments requiring in-lab approval, to ensure that students do not copy each other's solutions or at least that they understand what their code does.

3 QuickFeed Technical Details

This section gives a brief overview of QuickFeed's system architecture, as shown in Figure 2. QuickFeed is comprised of the QuickFeed server written in Go, Docker containers for running solution checkers, a SQLite database to store course, user, and other relevant information. In addition, we use the Envoy proxy to mediate external traffic. The QuickFeed server performs many tasks, among them (1) authenticate and authorize user accesses based on their course role, (2) to serve the React-based web application code to the student and teacher browsers, (3) to receive webhook events from GitHub when students or teachers modify a source code repository, (4) run solution checkers using Docker, (5) collect score and log output and record this in the database, and (6) to service data requests from the React-based frontend application to be displayed to users.

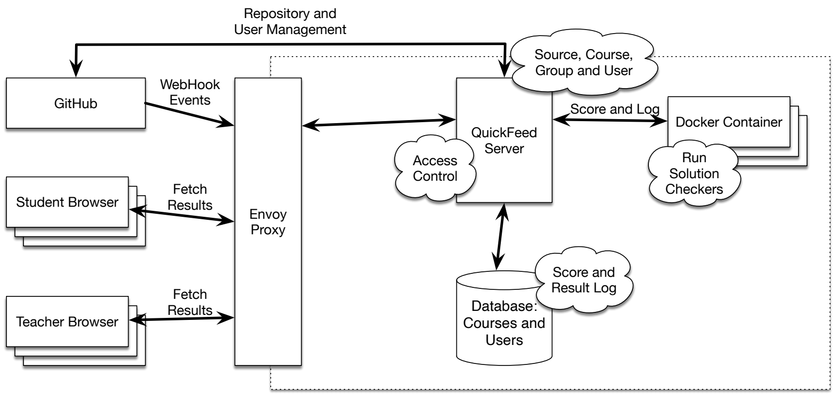


Fig. 2: QuickFeed's System Architecture.

4 Designing Assignments and Solution Checkers

In this section, we explain QuickFeed's approach to organizing assignments and solution checkers. QuickFeed uses the following repository structure within an organization on GitHub. These will be created automatically when a course is created using QuickFeed.

Repository	**Description**	**Access**
`info`	Holds information about the course	Public
`assignments`	Contains a folder for each assignment	Students, Teachers, QuickFeed
`username-labs`	One for each student	Student, Teachers, QuickFeed
`tests`	One folder for each assignment	Teachers, QuickFeed

The `assignments` repository has a separate folder for each assignment at the root level. Each assignment has a corresponding folder in the `tests` repository, that contains an `assignment.yml` file describing the assignment (see below). The `tests` folder also contain solution checkers for each assignment. The username-labs repository is initially empty, and the student must fetch template code from the course's assignments repository.

When QuickFeed receive a notification from GitHub that a student has made changes to their codebase, QuickFeed creates a Docker container that fetches the student's code and the solution checkers from the tests repository, and merges the content of the two repositories, and runs the solution checkers to test the student's code. The checkers produce log output that the student can later view in the web application. In addition, the checkers emit several JSON score objects that QuickFeed picks up, which are used to calculate the overall score of the assignment.

```
assignmentid: 1
title: "Introduction to Unix"
deadline: "2020-08-30T23:59:00"
autoapprove: true
scorelimit: 90
isgrouplab: false
```

```
tests-|
      |--- scripts
      |    |-- Dockerfile
      |    |-- run.sh
      |--- lab1
      |    |-- assignment.yml
```

As shown above, the tests repository of a course can provide their own Dockerfile and run.sh file. These are used to run solution checkers for the specific course, giving the necessary flexibility to support different course requirements, such as programming languages, tools, and compilers. We have used QuickFeed in courses using Java, C#, Python, and Go.

The (partial) code snippet in Figure 3 shows a simple solution checker in Go for an assignment where the students must implement a concurrent version of the popular fizz buzz game. The init() function adds the checker function to the set of checkers, along with the maximum score (len(fizzBuzzTests)) and weight (20). The TestFizzBuzz checker function obtains a score object sc, whose initial value is the max score. For each check whose result is incorrect, the score is decremented using the sc.Dec() method. The defer sc.Print(t) statement is executed at the end of the checker and emits a JSON encoded score object that QuickFeed can extract from the log output.

```go
func init() {
    scores.Add(TestFizzBuzz, len(fizzBuzzTests), 20)
}

func TestFizzBuzz(t *testing.T) {
    sc := scores.Max()
    defer sc.Print(t)
    for _, test := range fizzBuzzTests {
        gotResult := FizzBuzz(test.n)
        if diff := cmp.Diff(test.result, gotResult); diff != "" {
            sc.Dec()
            t.Errorf("fizzbuzz.FizzBuzz(%d): (-want +got):\n%s", test.n, diff)
        }
    }
}
```

Fig. 3: Example solution checker.