

Automated Grading of Programming Assignments with Continuous Integration and Docker

Dave Polansky, advised by Dr. Gudrun Socher

California Polytechnic State University, June 2017

1. Introduction	2
2. Analysis of Current Tools	2
2.1. File Transfer Tools	3
2.1.1. Student Perspective	3
2.1.2. Instructor Perspective	4
2.2. Automation with Bender	4
2.2.1. Student Perspective	4
2.2.2. Instructor Perspective	5
3. Related Work	6
4. Continuous Integration	6
5. Proof of Concept: grader-ci	8
5.1. Overview	8
5.2. Building	9
5.3. Grading	10
5.4. Technologies	10
5.5. Docker	10
5.6. Messaging Queue / Scaling Support	11
6. Evaluation	11
7. Summary and Outlook	13
8. References	13

1. Introduction

Academic courses that teach programming skills and theory help prepare students for real world problems and the software development industry. Although concepts such as knowledge of data structures, algorithms, and computational theory are fundamental to any Computer Science (CS) curriculum, students also develop software engineering (SE) skills and practices, such as testing and iterative development. As such, any means to improve a student's SE skills through the course's infrastructure, that is, without requiring any additional curriculum, should be considered by universities. This paper will analyze the potential for improving SE skills amongst students through the process of handing in and grading programming assignments.

Grading programming assignments manually is notoriously challenging and time consuming for the instructor or grader because it often requires reading code line by line to understand its behavior. Because manual inspection of code is often infeasible given large class sizes and limited time, many instructors implement tools to aid in grading assignments. Many of these tools focus on correctness, such as whether a student's submission compiles and produces correct output against given inputs. Although these solutions are adequate in terms of correctness, they raise the following concerns:

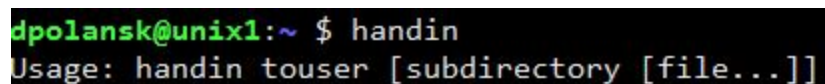
- Is it designed to encourage students to test their own code and iteratively develop solutions?
- Is it designed to prepare students for deploying software in the industry?
- Is it powerful enough to automate compiling, building, running, and testing?
- Is the solution flexible enough to meet the specifications of any programming course?

The simple process of handing in and grading programming assignments, which could be viewed as tedious to professors, has the potential to not only relieve instructors of work, but also influence the development of SE skills and practices. This paper provides a proof-of-concept container-based alternative to this process as well as commentary on tools being used in practice at Cal Poly.

2. Analysis of Current Tools

In order to understand the potential benefit of developing a new tool for handing-in and grading programming assignments, the technologies being used at Cal Poly will be analyzed based the criteria mentioned above.

2.1. File Transfer Tools



```
dpolansk@unix1:~ $ handin
Usage: handin touser [subdirectory [file...]]
```

Figure 1: Running the “handin”(Scheftic, 2005) command-line program from a terminal.

One of the most commonly used techniques for handing-in and grading assignments is to copy files from a student’s machine to a university server, or to copy files between accounts within the server. The mechanism for this technique is generally a command-line application that the student runs through a terminal, and these applications will be referred to as “file transfer tools”.

Two file transfer tools used at Cal Poly are “turnin” and “handin” (Scheftic, 2005), and they accomplish the same task of copying a student’s code to a directory which the instructor or grader has access to. Figure 1 shows a student invoking the handin program from a terminal, which prints instructions for submitting files from the student’s directory to the grader’s directory. Through the use of file transfer tools, the burden of grading assignments is placed entirely on the instructor because instructors must develop their own software to automate grading tasks, such as running test cases against every student’s code, recording output and results, and emailing students.

2.1.1. Student Perspective

File transfer tools eliminate the potential for a tool to provide students with adequate feedback about the accuracy of their assignments and whether it runs on the grader’s machine. First, consider the common use case of developing code in an environment that does not match the grader’s environment. For example, a student may be developing on a 32-bit machine, yet the grader uses a 64-bit machine. To ensure that an assignment runs correctly on the grader’s machine, the student not only must have access to the grader’s machine (or set up their own virtual machine that matches the grader’s environment), but they also incur tremendous overhead in development time whenever they handin their assignment. The student must manually transfer their files to the grader’s machine, run their test cases on the grader’s machine to ensure it runs correctly, and then finally use the file transfer tool to copy the files to the instructor’s grading directory.

In addition to the overhead of manually checking whether their code runs on the grader’s machine every time they submit their code, file transfer tools do not run a student’s code against test cases. Not only does this prevent the student from receiving feedback about the accuracy of their submission in terms of passing the instructor’s test cases, but it also results in poor

turnaround-time for receiving a grade for the assignment because their code is not run until the instructor manually grades the assignments.

2.1.2. Instructor Perspective

File transfer tools limit instructors to a “closed-door” policy of grading, which means that no feedback can be given to students about their submissions until after the deadline for the assignment has passed. Furthermore, these tools place the burden of testing, grading, and sending results to students on instructors or graders. Many professors at Cal Poly hire Teacher Aides whose sole responsibility is to write grading scripts which run test cases for each student’s submission and email students their results. Both of these consequences of using file transfer tools simply add more work for the instructor or the grader because the tool being used is not flexible enough to automate these processes for every assignment.

2.2. Automation with Bender

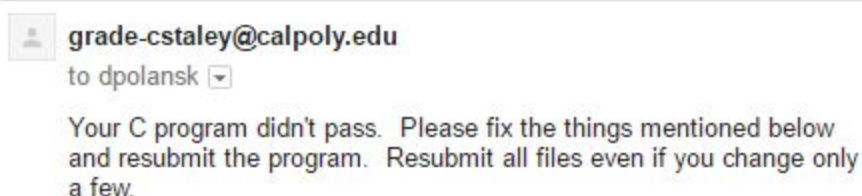


Figure 2: An automated email from Bender (Staley, n.d.) about the status of an assignment submission.

Bender is the “grading robot” developed by Professor Clint Staley which aims to improve upon file transfer tools by automating several of the instructor’s tasks when grading assignments to provide rapid feedback for students (Staley, n.d.). After a student submits an assignment using the file transfer tool “turnin”, Bender periodically runs the submission against the instructor’s test cases and sends an email to the student with results. Figure 2 shows a response email for a student’s assignment submission, which contains any compilation errors or test failures within the student’s code. The Bender program runs student submissions on a schedule, “Bender runs at 1a, 7a, 1p, and 7p, every day including weekends”, in order to prevent students from using the program to debug their code.

2.2.1. Student Perspective

The use of an automated grading tool improves the student’s development by providing them with feedback about their submission. It is unclear whether Bender was purposely designed to only grade submissions at specific times in the day in order to challenge students by preventing them from frequently checking their submissions. If we assume this is not a limitation of Bender itself, it should be noted that Bender provides an adequate solution to handing in assignments for basic use cases.

Bender's rigidity in terms of scheduling and secrecy in terms of output sent back to the student could be suitable for certain academic use cases. However, Bender misses the opportunity to provide students with hands-on experience with tools students will be using in industry, such as developing with version control systems such as Git, which will be discussed further in the Continuous Integration section.

2.2.2. Instructor Perspective

As mentioned above, Bender provides an automated solution to an instructor's tasks when grading assignments, including running test cases and sending a report back to the student about their submission. Although Bender is technologically sufficient in terms of compiling and running C code against test cases for the course it was used in, Systems Programming (CPE 357), Bender would need to support several features for use throughout the entire department and in other universities, such as the following:

Can the tool adapt to any of the dependencies required by a student's code? Consider the scenario where a student's code relies on third-party packages, such as a Java project using the Maven dependency and build management tool, which downloads third-party packages from Maven's central repository. Bender is capable of compiling and running C programs, which only serves a small subset of the possible assignments and projects instructors may want to incorporate into their curriculum.

Additionally, since Bender relies on the dependencies and architecture of the environment it is running in, the tool is unable to accommodate programs that require a specific environment. For instance, if a student's project relies on a library that only runs on Windows when Bender itself is running on Linux, an ideal tool would need to set up a virtual environment to replicate the student's development environment.

It also must be considered whether the tool dynamically scale to accommodate the demands of classes and assignments of arbitrary size. Consider a large lecture course with 200 students in which each student is trying to submit and check their code before an assignment's deadline using Bender. Since Bender was developed for use in a single class at fixed scheduling intervals, Bender lacks any means to concurrently run many builds across an arbitrary number of machines.

Furthermore, in order to scale in terms of building a single tool to suit the needs of a university, the means of automating build tasks must be language independent and must not involve recompiling or hard-coding assignment-specific details into the tool. For instance, if the tasks of

compiling and diff-ing outputs from C programs are hard-coded into Bender, then the tool must be modified for use in other situations.

3. Related Work

One paper discussed the implementation and theory of an automated online grading system called the “Online Judge” (Cheang, 2003, p. 122). The paper discusses pitfalls with human grading, such as inconsistency of grading, judging correctness, and time spent manually reading through or running code against test cases. The paper commented on the potential for malicious programs to cause harm to the host system, to which this danger was alleviated with the use of the **chroot** (Debian Wiki, 2016) UNIX command, which changes the root directory for a running process. In contrast, this project utilizes Docker to achieve the same effect, through the use of Linux namespaces (Kerrisk, 2017). Overall, the paper noted the benefits of instantaneous feedback for students and the ability to grade quickly and fairly. Furthermore, it warns that this system relies on stringent specifications, that feedback to students is limited, and that certain aspects of code style should still be checked manually, such as maintainability.

Another paper discusses the potential for an automated grading tool to encourage students to write tests (Edwards, 2003, p. 2). It discusses how an automated grading infrastructure allows classes to check and enforce code coverage on each submission, effectively forcing students to develop in a pseudo-TDD style. Its implementation used the Web-CAT (The Web-CAT Community, 2017) tool along with static analysis tools to judge correctness and style, while enforcing testing standards with code coverage tools. The paper notes that a class of 59 students preferred developing in TDD style compared to previous classes.

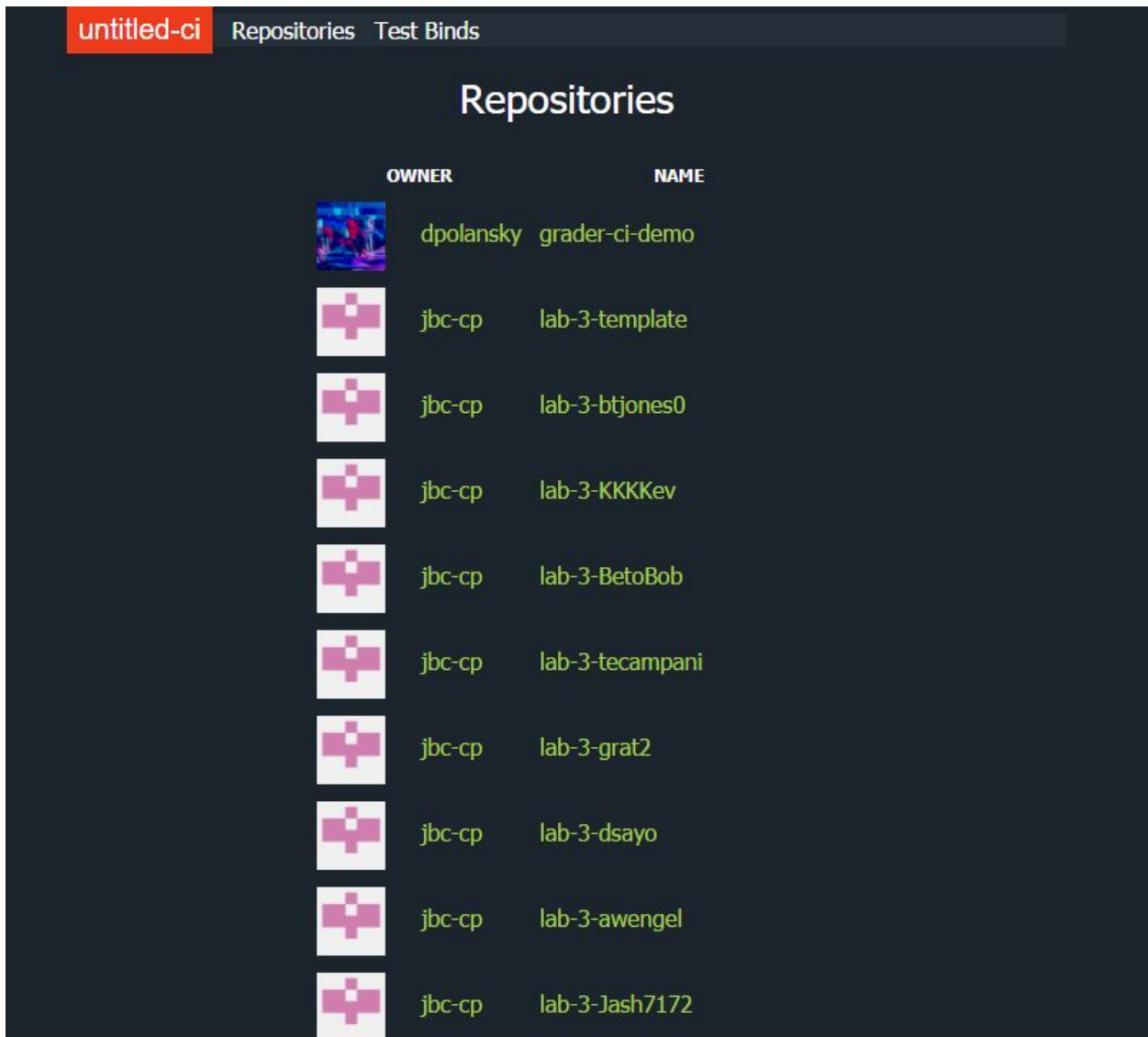
Furthermore, one paper built a REST API that interfaces with the Intelligent Tutoring System (ITS), a pre-existing system used for grading programming assignments (Parihar, 2015, p. 5). The paper discusses several methods for automatically computing scores for students’ assignments, based on factors such as program run-time, number of correct compilations, and number of test cases passed. This highlights the need for a grading tool to not only automatically assign grades to students, but also collect data and metrics about a submission’s performance. However, judging many of these criteria without human input would rely on language and framework specific code, which could defeat the purpose of building a dependency-abstracted solution using Docker. In addition, this solution relies on an external service to handle grading of assignments, which could be cumbersome for small use cases.

4. Continuous Integration

Continuous integration (CI) is a software engineering practice of automatically testing and deploying code when changes are made (Fowler, 2006). The practice originates from extreme programming as one of its twelve practices. In the software development industry, the use of CI tools are used to enable multiple developers to iterate on a given project and make changes to the source repository with rapid feedback about whether the build succeeded. If a developer commits a change and the build fails, the CI system could prevent it from being deployed and notify the developer of the failure. This feedback loop allows developers to maintain a stable working copy of code and increases visibility of errors in code. Instead of pushing thousands of lines of code to production and hoping it works despite the possibility of other developers making changes to the code, developers with CI systems can frequently push small changes and remain confident in the health of the system.

This approach to software development is precisely the methodology that CS/SE instructors teach to students: testing early and often, problem solving in small increments, etc. Although students have a responsibility to test their own code, a CI server will enable students to learn the same development techniques that they will use in the industry. In contrast, file sharing tools that simply upload a student's files without any building and testing are equivalent to software companies that blindly push code to production servers.

5. Proof of Concept: grader-ci













	OWNER	NAME
	dpolansky	grader-ci-demo
	jbc-cp	lab-3-template
	jbc-cp	lab-3-btjones0
	jbc-cp	lab-3-KKKKev
	jbc-cp	lab-3-BetoBob
	jbc-cp	lab-3-tecampani
	jbc-cp	lab-3-grat2
	jbc-cp	lab-3-dsayo
	jbc-cp	lab-3-awengel
	jbc-cp	lab-3-Jash7172

Figure 3: A table view of all repositories being graded by the system.

5.1. Overview

This project aims to explore the feasibility of building and using a continuous integration server as a replacement for handin tools in an academic setting, developed using Go. The system contains two core processes: the backend, and the worker.

The system fundamentally relies on repository hosting services, such as Github, which provides students with a means to host their code and interact with their code with version control

systems, such as Git or SVN. These services provide a webhook functionality which grader-ci uses to receive notifications when a student has made changes to their code. The webhook functionality from these services is what allows the system to automatically test and grade repositories.

The backend process is a web server which serves a REST API for fetching and storing data about assignments (which are abstracted as builds) and handling webhooks from repository hosting services. The server also interacts with a distributed messaging queue to communicate with an arbitrary number of worker processes to run and collect results for building and testing assignments.

The worker process connects to a distributed messaging queue and waits for build jobs from the backend server. It builds, tests, and “grades” assignments in containerized environments with the assignment’s dependencies and sends status updates to the backend through the messaging queue. Worker processes can be created as needed to handle larger class sizes without any additional configuration.

5.2. Building

Students create a git repository for their assignment and a configuration file specifying any dependencies or scripts to be run when a build is run. The workflow for a student’s basic use case is as follows:

1. Student creates a git repository and a configuration file.
2. Student commits their code to the repository and pushes the repository to a hosting service, such as Github.
3. The hosting service’s webhook creates an HTTP request to the backend notifying it about the new commit.
4. The backend handles the webhook, records its data in a database, and sends a build job to a distributed messaging queue for a worker process to handle.
5. A worker process receives the build job from the queue, clones the student’s repository, and creates a Docker container with the project’s dependencies and target environment.
6. The worker runs the build in the container and sends status updates to the backend server.

At any time during the build process, the frontend displays the build’s status and log output, along with all previously run builds and repositories. Figure 3 shows the root page of the frontend, which displays a listing of all the repositories that have been tested or graded by the system.

5.3. Grading

In the same way that students develop their assignments using Git repositories, the instructors automate testing and grading through Git repositories as well. When testing the assignment against the instructor's test cases, the student creates a "binding" through the web interface from their source repository to the instructor's grading repository. A grading repository contains test files and/or source code that will be merged into the source repository when a build is triggered. This means that when a student makes a commit to the source repo, grader-ci will test it in combination with the grader's repository.

For example, a student's repository might look like:

- `fizzbuzz.go`
- `.ci.yml`

and the grading repository might look like:

- `fizzbuzz_test.go`
- `.ci.yml`

When the source repository is being built, `fizzbuzz.go` will be tested with the grader's `fizzbuzz_test.go`. This format provides the instructor with the flexibility to specify exactly how they want to test their students' code because they simply create a repository and configuration file that describes what to do with each assignment's source files.

5.4. Technologies

5.5. Docker

The project uses Docker containers to provide the flexibility of supporting any program dependencies as well as an execution environment for custom scripts. Docker is an open-source platform for packaging software into "images", which can be run in isolated environments (Docker, 2017). Docker provides immense utility for a CI system because images can be defined to contain all of the dependencies necessary to run a given assignment without having to fetch the dependencies at build time for every assignment. This means that the same build image can be used for every student in a particular class, which drastically reduces build times.

Isolated environments are necessary when building a CI system for the purposes of widespread academic use because each class can have varying project specifications. For example, if one class relies on Python 2.X and another relies on Python 3.X, a single grading server would need to implement logic to switch between the two versions of Python. Although trivial, the use of Docker images allows the instructor to treat the build environment as a blank canvas and provide

only the dependencies necessary for the project. Additionally, the CI server itself does not need to have any business logic related to building and running code as long as it knows how to create and run Docker containers. This makes it possible for several instructors to use the same tool without having to recompile to support special use cases.

5.6. Messaging Queue / Scaling Support

Although this project in particular makes use of a distributed messaging queue, RabbitMQ (CloudAMQP, 2015), to send build tasks to worker processes, any CI solution that is intended for use by multiple classes should have support for multiple processes. The same result could be accomplished with a load balancer that reverse-proxies several instances of the CI service.

In this project, RabbitMQ is used as the communication link between the backend server, which hosts a REST API and communicates with a database, and worker processes, which know how to run builds in Docker containers. With this design, worker processes can be created and destroyed as needed to handle build traffic, such as when a class is approaching the due date for an assignment.

6. Evaluation

To test the system and demonstrate its ability to integrate seamlessly into an introductory CS programming course, a professor granted access to over 300 student assignment repositories. This particular class had already started using Github to host student repositories prior to agreeing to make them available for demo purposes, which made it simple to integrate with grader-ci because it relies on interacting with git repositories.

The existing grading solution in use for the class was a cron job that fetched all students' repositories, and then executed a script to run test cases, collect results, etc. Since grader-ci supports user defined build scripts, integrating the entire pre-existing script into the grader-ci structure was a possibility. Since this demo was not intended to be an active replacement for the class, only the tests were run, ignoring potential tasks such as emailing students and grading based on number of tests failed.



Figure 4: The files contained in the grading repository.

To get started, a grading repository was created containing a configuration file (.ci.yml), and the Python test cases that would need to be run with the student's source code. Figure 4 shows the .ci.yml configuration file and two python files containing the instructor's test cases, lab3tests_al.py and lab3tests_ll.py, which test the student's implementation of ArrayList and LinkedList data structures.

The configuration file, .ci.yml, contains the following yaml:

```
language: python3
script:
  - python lab3tests_al.py
  - python lab3tests_ll.py
```

The language value refers to the Docker image that will be used when running the build, and the script refers to commands that will be run when the student's assignment repository is merged with the grading repository. Although the system would work immediately at this point if webhooks were set up for each assignment repository, a fake webhook was created for each repository to trigger a build.



Figure 5: A page that shows details of a student's graded build.

Figure 5 shows the output of one student's graded build, which contains the output of the build and a link to the grading repository. The web interface provides students with easily accessible information about each build which helps the student understand how their code interacts with the grading environment.

7. Summary and Outlook

The process of analyzing existing grading and delivery tools has shown the potential benefit of introducing an automated container-based CI solution. Pre-existing solutions, such as file transfer tools, lack the ability to automate an instructor's grading tasks and miss the opportunity to provide students with feedback about their code. Furthermore, automated tools such as the Bender Program are not flexible enough to grade assignments of varying languages because they lack a container-based infrastructure, which prevents the need to recompile the tool to support new dependencies. In addition, many of these tools are not designed to scale with large class sizes or across an entire department, which makes them infeasible for widespread adoption throughout a university.

The grader-ci proof of concept demonstrates how a single system with distributed worker processes and a container-based infrastructure can be used to meet the demands of any class size or assignment specifications. Furthermore, by implementing the grading tool as a CI server for a class, students are provided with experience for tools and methodologies they will use in their software engineering career, such as working with version control systems and analyzing test coverage. This proof of concept has also raised questions about how a tool could automatically assign grades based on build output (Parihar, 2015) or how data could be collected to provide instructors with feedback about how their students are performing. Moving forward, the process of implementing this grading system relies on educating academic institutions about how their pre-existing solutions could be improved and how open-source software tools, such as git, Docker, and RabbitMQ can be implemented to solve logistical problems.

8. References

Cheang, B., Kurnia, A., Lim, A. and Oon, W.C., 2003. On automated grading of programming assignments in an academic institution. *Computers & Education*, 41(2), pp.121-131.

Part 1: RabbitMQ for beginners - What is RabbitMQ?, 2015, CloudAMQP, viewed 4 June 2017, <<https://www.cloudamqp.com/blog/2015-05-18-part1-rabbitmq-for-beginners-what-is-rabbitmq.html>>

‘chroot’, 2016, *Debian Wiki*, viewed 4 June 2017, <<https://wiki.debian.org/chroot>>

What is Docker, Docker, viewed 4 June 2017, <<https://www.docker.com/what-docker>>

Edwards, S.H., 2003, October. Teaching software testing: automatic grading meets test-first coding. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* (pp. 318-319). ACM.

Fowler, M, 2006. *Continuous Integration*”, viewed 4 June 2017, <<https://martinfowler.com/articles/continuousIntegration.html>>

Kerrisk, M, 2017, *namespaces - overview of Linux namespaces*, Linux man-pages Project, viewed 4 June 2017, <<http://man7.org/linux/man-pages/man7/namespaces.7.html>>

Parihar, S., 2015. *Automated Grading Tool for Introductory Programming* (Doctoral dissertation, INDIAN INSTITUTE OF TECHNOLOGY KANPUR).

Scheftic, C, 2005, *The “handin” process*, viewed 4 June 2017, <<http://users.csc.calpoly.edu/~buckalew/handinRef.html>>

Staley, C (n.d.), *Guide to Bender and Turning*, viewed 4 June 2017, <<http://users.csc.calpoly.edu/~cstaley/General/TurninAndBender.html>>

Web-CAT, The Web-CAT Community, viewed 4 June 2017 <<http://web-cat.org/group/web-cat>>