1. **Setting up for TDD analysis**

**Environment set-up**

Code is located on Github at <https://github.com/sah0017/TDDGitLogAnalysisCode.git>.

* Required installations
  + Jsonpickle - The git analysis uses jsonpickle. Instructions for its installation can be found here: https://jsonpickle.github.io/.
  + Coverage - For the code coverage analysis to work, you will have to install the coverage module. Instructions for installation are found here: <https://coverage.readthedocs.io/en/coverage-4.1/install.html>.

In development, the platform was Windows and the initial path for data was g:\git\6700Spring16\. Under the semester-named folder, assignments each had a separate folder, CA01, CA02, etc. Submissions were unzipped into a submissions folder under the assignment-named folder. Generically, the path would be

Root:\Home directory\Semester directory\Assignment directory\**submissions**\student submission directories…. The path names can change as needed for different environments.

**TDDanalysis.cfg file**

To support portability, the system has a TDDanalysis.cfg file, which is located in the same directory with the source code. In the file, there are five sections: Location, TA Test Case, HTML Location, Assignments, and Due Dates.

# Location section

This section contains the elements Root, Home, Semester, Assignment, and Name Path Depth. These elements help the code find where you have located the student submissions. For the code to work correctly, each of these must contain a valid part of the path to the submissions. In development, the config file looks like this:

|  |  |
| --- | --- |
| Windows example | Mac example |
| [Location] | [Location] |
| Root: g: | Root: Users |
| Home: git | Home: shammond/GoogleDrive |
| Semester: 6700Spring16 | Semester: 6700Spring17 |
| Assignment: CA05 | Assignment: Assignment5 |
| Name Path Depth: 7 | Name Path Depth: 7 |

Name Path Depth tells the program how deep into the directory tree it will find student file directories. This will allow for some flexibility as to the depth of the overall directory tree structure.

# TA Test Case Location section

This section deals with TA Test Cases. If a student copied the TA Test Cases into their own test case files, he/she doesn’t receive test Lines of Code credit for adding original test case lines. In the config file, this section looks like this:

[TA Test Case Location]

Test Directory: TATests

The analysis code will look for TA Test Code in this directory, which should be placed underneath the Semester directory (e.g., Root:\Home\Semester\TATests). Under the TATests folder, it does not matter if the test files are all in the same directory or in multiple directories. The code will read the contents of the TA Test Code and compare it to test code contents for each student.

# HTML Location section

This section deals with an HTML Code Coverage report that can be provided for each individual. If all of a student’s test code successfully passes, the Code Coverage process can generate an HTML report showing where the coverage is lacking. This section in the TDDanalysis.cfg file will allow you to define the location to place the many HTML files generated to support this report. In development, the elements used the same location, but you may wish to set up a different directory structure that the students can access and place the files there. This section will allow you to define the location to place the files. Students can be instructed to go there and click the index.html file to view their report. In the config file, this section looks like this:

[HTML Location]

Root: g:

Home: git

Semester: 6700Spring16

Assignment: CA05

The code will create a directory under this path with the student’s submission name and place the html and supporting files in that directory.

# Assignments section

This section gives the system information about the assignments. NumberOfAssignments tells how many coding assignments are represented in the student submission’s git files. Typically assignments begin with a BaseName (in the past there have been bases such as CA or Assignment). The BaseName element allows for variations in the Assignment names from one semester to the next. FirstTDDAssignment just lets the system know which is the first assignment that contains code developed using the TDD process.

[Assignments]

NumberOfAssignments: 4

BaseName: Assignment

FirstTDDAssignment: 3

# Due Dates section

This section lets the system know when the due dates were for each assignment. To allow for late submissions, I typically set the due dates at least a week past the published due date. The system uses this to distinguish between assignments. The assignment name should be the base name and the assignment number concatenated together.

[Due Dates]

Assignment3: 2017, 3, 19

Assignment4: 2017, 3, 27

Assignment5: 2017, 4, 17

Assignment6: 2017, 5, 2

**Presence of TA Test Cases**

If there are TA Test Cases, they should be placed in the TA Test Location as mentioned above. Once they are placed there, you should run TATestCase. This creates a json file that contains the names of all the individual tests, plus the number of lines of code in each test. **TATestCase should be run every time new TA Test Cases are added to the TA Test directory.**

**Utility to download .git files**

Mass Clone is a utility to download multiple files from Github Classroom. It can be downloaded using the following command: git clone <https://github.com/konzy/mass_clone>. Windows-based machines require Dygwin to run Mass Clone. Modify the file named clone\_all\_helper\_example.sh, enter the organization as SoftwareProcess, use your username, and the https protocol.  From the mass\_clone subdirectory, run ./clone\_all\_helper\_example.sh on the current assignment using the unique identifier of the assignment.  Make sure the upper/lower case is correct.

The github files will all be downloaded into a directory of the same name as the assignment. **You will need to move all the files under a folder named submissions in your directory tree.**

1. **TDD/TPP Analysis – runGitfileAnalysis.py**

The program runGitfileAnalysis has three phases.

**Phase 1**

Have you created the formatted git files? (y/n)

If you answer y, runGitFileAnalysis will read through the git repositories of every student submission and create a specially formatted, text-based git file that is used to analyze their TDD performance. These files are placed under the Root:\Home\Semester\Assignment directory with a .gitdata extension, one per student with the directory name from their submission as the file name.

The system will print out to the console the student directory name as it converts the git repository. The following are issues that may arise:

|  |  |  |
| --- | --- | --- |
| Possible error messages/issues | Meaning | Action |
| fatal: not a git repository (or any of the parent directories): .git | This indicates that they created a .git directory, but didn’t really create a git repository | Instruct student to create a valid git repository on the next assignment |
| The code does not create a .gitdata file for a particular student | The student didn’t create a git repository | Instruct student to create a valid git repository on the next assignment |

If you have already created the formatted git file but need to re-run the analysis, you can skip the formatted git file creation process, which does take some time. When you encounter the question ‘Have you created the formatted git files? (y/n)’, you can answer y and it will skip that step.

**Phase 2**

After the .gitdata files are created, runGitfileAnalysis will perform the second phase, which is the TDD analysis of all the .gitdata files. This will create two files per student, one with a .json extension and one with a .gitout extension, both with the student directory name as the file name. These files are found in the Root:\Home\Semester\Assignment directory, along with the .gitdata files.

The .json file is used to create the Analysis Report in the final phase. It also allows you to re-run the Analysis Report while skipping the first two phases in runGitfileAnalysis.

The .gitout file provides an individual report per student of the Assignments contained in their git file, and it contains two sections. This report may be run for all of the students’ assignments, or for an individual assignment. You will be prompted with the question ‘**Report on all assignments or just 1 (type 'all' or assignment Name)’** If you enter the assignment name, it should begin with assignment and a number with no space, e.g. assignment4.

The first section of the .gitout file is sorted by Assignments, and then Commits within the Assignment, and it shows statistics and transformations by commit with Assignment totals at the end of each Assignment. The second section lists each of the files created by the student and displays commit statistics and methods added/modified to that file per commit. These reports can be provided to the students if we believe the feedback is valuable to them.

If you just need to re-run the overall Assignment Report, you can skip the first two steps. The system will prompt with the questions: Have you created the formatted git files? (y/n) and Have you analyzed the formatted git files? (y/n).Just answer y to both questions and it will skip those steps.

**Phase 3**

The final phase is the creation of an Assignment Report. It contains statistics by student and by assignment, with totals at the bottom. This file will be written in the Root:\Home\Semester directory with a name of Report concatenated with the Assignment Name with an extension of .csv (ex. ReportAssignment1.csv). This is a tab-separated text file which is usually imported into Excel for formatting.

Things to check in the Assignment Report include the following:

|  |  |  |
| --- | --- | --- |
| Possible error messages/issues | Meaning | Action |
| Student has very few total Number of Commits | Student is not following the TDD pattern or is not committing to git after each TDD step. | Instruct student to follow the prescribed TDD process and use the plug-in buttons to commit and run their code. |
| Student has a large number of commits, but they are all classified as Other | Student is committing to the .git file but isn’t using the plug-in buttons to do their commits. | Instruct student to follow the prescribed TDD process and use the plug-in buttons to commit and run their code. |
| Student has a large number in the Avg Lines per Commit column | Student is not following the TDD pattern. | Instruct the student that TDD should be done in small increments, which would result in a small number for the average lines per commit. |

1. **Code Coverage Analysis**

There are two or three steps to performing Code Coverage Analysis, depending on how well the students followed instructions in writing their submissions.

1. **Run GetDirectoryList.py**

This program will create a directory list of the submissions to be analyzed. The directory listing will be written into the directory with the Python code. It will create this list for the Assignment element set in the TDDanalysis.cfg file.

1. **Run CodeCovAnalysis shell/batch file from CMD/Terminal**

CodeCovAnalysis.bat is a Windows-based batch file that is located in the same directory as the Python code. CodeCovAnalysis.sh is a Mac-based shell script that does the same thing as CodeCovAnalysis.bat. It contains the following commands:

set arg1=%1

set arg2=%2

FOR /F %%i IN (%arg1%.dirlist) DO CodeCoverage %%i %arg1% $arg2%

CodeCovAnalysis must be run from the command line (CMD for Windows, Terminal for Mac). For Windows, it would be called like this:

CodeCovAnalysis Assignment5.dirList yes

For Mac, before running the script for the first time, you may need to run the following command: chmod u+x /CodeCovAnalysis.sh

Then you call the shell script like this:

./CodeCovAnalysis.sh Assignment5.dirList no

When you run CodeCovAnalysis, you are passing it two arguments. The first is the assignment name being analyzed, and the second tells it whether it should generate the HTML Code Coverage report that is available (i.e., CodeCovAnalysis CA05 yes). If you choose to generate the HTML report, be sure that the appropriate section is present in the TDDanalysis.cfg file described above.

The shell/batch file will loop through the appropriate directory listing file (created by running GetDirectoryList.py) and call the Python file CodeCoverage, passing it the appropriate file path and assignment to evaluate.

CodeCoverage will exercise each student’s test code against their own production code. Results of the Code Coverage analysis will be written to a text file found in the Root:\Home\Semester directory and named using the Assignment Name with an extension of .CCreport (ex. CA01.CCreport). For each submission, a line is generated showing the Student submission name and path, as well as a list of the test file names imported and executed. If the student used any print statements in their test or production code, these statements will be captured and recorded in the report as well.

Because the report file is opened and closed for each student submission, the information is appended to the file. **If you need to re-run CodeCovAnalaysis.bat, delete the previous version of the code coverage report. If you don’t, the results of the latest run will be found at the bottom of the report.**

For each student submission analyzed, there are three potential results, and the results will be displayed after the list of test names.

In the first scenario, all of the test code completes successfully, and a code coverage report will be generated and entered into the report. If the batch file was called requesting the HTML report, the appropriate directory will be created along the path described in TDDanalysis.cfg and the files will be saved to that location.

Example:

Student submission and path: g:\git\6700Spring16\CA05\submissions\zhuqi\_3232019\_74848028\_QiZhuCA05\COMP6700CA05\CA05\

Test File Names

Longitudetest\_\_init\_\_Number of tests run: 14

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Stmts | Miss | Branch | BrPart | Cover |
| G:\git\6700Spring16\CA05\submissions\zhuqi\_3232019\_74848028\_QiZhuCA05\COMP6700CA05\CA05\prod\Longitude.py | 25 | 1 | 10 | 1 | 94% |
| Longitudetest.py | 65 | 0 | 0 | 0 | 100% |
| \_\_init\_\_.py | 0 | 0 | 0 | 0 | 100% |
| TOTAL | 90 | 1 | 10 | 1 | 98% |

A second scenario is that the student’s test code failed to complete successfully. If this occurs, the exceptions thrown by the test code are captured and written into the report. The test code can fail for any number of reasons, for example, syntax errors, inexact matches between test code asserts and production code return values, uncaught exceptions, etc. Students should be informed that their test code must pass when run against their own production code.

Example:

Student submission and path: g:\git\6700Spring16\CA05\submissions\almohaishimoayad\_3221348\_74842094\_mha0012CA05\softwareProcess\SoftwareProcess\Assignment\

Test File Names

FixTestLatitudeTestLongitudeTest2SightingTestStarCatalogSample\_\_init\_\_Traceback (most recent call last):

File "G:\git\6700test\6700\tppAnalysis\CodeCoverage.py", line 76, in analyzeCodeCoverage

load = myTestLoader.loadTestsFromNames(moduleTestNames)

File "c:\Python27\Lib\unittest\loader.py", line 130, in loadTestsFromNames

suites = [self.loadTestsFromName(name, module) for name in names]

File "c:\Python27\Lib\unittest\loader.py", line 91, in loadTestsFromName

module = \_\_import\_\_('.'.join(parts\_copy))

File "g:\git\6700Spring16\CA05\submissions\almohaishimoayad\_3221348\_74842094\_mha0012CA05\softwareProcess\SoftwareProcess\Assignment\\test\StarCatalogSample.py", line 46, in <module>

starCount = stars.loadCatalog(starFile="Sao.Txt")

File "g:\git\6700Spring16\CA05\submissions\almohaishimoayad\_3221348\_74842094\_mha0012CA05\softwareProcess\SoftwareProcess\Assignment\\prod\StarCatalog.py", line 91, in loadCatalog

raise ValueError("StarCatalog.loadCatalog: No such a file exist.")

ValueError: StarCatalog.loadCatalog: No such a file exist.

A third potential scenario is that the student puts something into their import statement that will not work in a different environment. This could include putting their dotted module path in the import statement, i.e. import Assignment.prod.Fix. A small utility named ScrubTestFiles will correct this import issue. First advise the students to remove the dotted notation from their import statements for future assignments. But to continue the analysis, run ScrubTestFiles, and then re-run CodeCovAnalysis.

Other import errors are not addressed by this utility. For instance, some students may hard-code a path into their import statements, i.e. C:\myCode\prod. These students should be penalized and instructed to remove the hard-coded paths.

Here is an example that has an import issue that does not include the prod dotted notation scenario:

Student submission and path: g:\git\6700Spring16\CA05\submissions\carrolljohn\_648283\_74766042\_jcc0044\_CA05\SoftwareProcess\SoftwareProcess\Assignment\

Test File Names

FixTestLatitudeTestLongitudeTestSCDriverTestingSightingTestSightingTest5StarCatalogSampleStarCatalogTestTraceback (most recent call last):

File "G:\git\6700test\6700\tppAnalysis\CodeCoverage.py", line 76, in analyzeCodeCoverage

load = myTestLoader.loadTestsFromNames(moduleTestNames)

File "c:\Python27\Lib\unittest\loader.py", line 130, in loadTestsFromNames

suites = [self.loadTestsFromName(name, module) for name in names]

File "c:\Python27\Lib\unittest\loader.py", line 91, in loadTestsFromName

module = \_\_import\_\_('.'.join(parts\_copy))

File "g:\git\6700Spring16\CA05\submissions\carrolljohn\_648283\_74766042\_jcc0044\_CA05\SoftwareProcess\SoftwareProcess\Assignment\\test\SCDriverTesting.py", line 33, in <module>

import StarCatalog as StarCatalog

File "g:\git\6700Spring16\CA05\submissions\carrolljohn\_648283\_74766042\_jcc0044\_CA05\SoftwareProcess\SoftwareProcess\Assignment\\prod\StarCatalog.py", line 32, in <module>

from Assignment.sandbox.FilePathExist import starFile

ImportError: No module named sandbox.FilePathExist

**1a. If necessary, run ScrubTestFiles.py**

This utility will remove prod-related dotted notation additions to the import statements in the students’ files, if they used it. It will loop through all of the submissions and scrub all test and production files.

**Final Code Coverage Summary Report**

A final summary report of the code coverage results is created at Root:\Home\Semester\Assignment, and is named <Assignment>.cvgrpt. This is a text-based, tab-delimited report that will list the student’s name, and one of three summarized results. It will either show a code coverage percentage, or one of two error messages: “Run Error” (indicating the code failed its own test cases) or “Import Error/Didn’t get to student’s tests”.

You can examine the code coverage report first to see which students’ code had issues, then consult the detailed .CCReport to indicate what the specific problems were with each student.

Output Files

Under each assignment, each student should have 3 result files.

.gitdata is a dump of the git log in oldest to newest order.

.gitout is output that comes out of runGitFileAnalysis. It contains the results of the TDD analysis. It has 2 sections per assignment:

* 1. The number of TDD cycles, as defined by a Red Light followed by a Green Light. Then a listing showing where the student used consecutive commits of the same type.
  2. A listing of all commits, giving the commit type and statistics about changes to the file(s) in that commit. There is a summary at the end of that section showing Total test/prod LOC added and deleted, and the ratio of test to prod code.

.json is a dump of the raw data created by the analysis. This is used to re-run analysis without having to do the first step under RunGitFileAnalysis.

At the root of the semester directory, there are 3 files per student:

.csv file for each student. It contains Process Conformance information, including an evaluation of the validity of their Red and Green Lights, as well as TDD and TPP conformance.

.cvg contains code coverage data for the specified assignment per student

The root of the semester directory also contains a .CCReport, which is a code coverage report. It will also contain any errors the students encountered with their test code.

Definitions of the columns in the Process Conformance columns:

| Phrase used | Definition |
| --- | --- |
| **Invalid Red** | Added prod code OR deleted prod code OR created prod file |
| **Red Light Validity Ratio** | Total RL - Invalid RL / Total RL |
| Invalid Green | Added test code OR Deleted test code OR Created test file |
| Green Light Validity Ratio | Total GL - Invalid GL / Total GL |
| TPP Conformance | Nbr of commits - Commits with too many trans / Nbr of commits |
| TDD Cycle | Begins with Red Light, contains a Green Light |
| Valid TDD Cycle | One transformation per commit, commits contain valid Red and valid Green Lights |
| Ideal Number of Cycles | Number of transformations performed + penalty values for anti-transformations |
| Current anti-transformations identified (+1 penalty unless noted otherwise) | Straight to Constant (no Null first)  Straight to Variable (no Null or Constant first - +2 penalty because they skipped 2 steps)  Straight to While (No If first) |