PUNIT USER MANUAL

School of Computing
Macquarie University, Balaclava Rd,
Macquarie Park NSW 2109

1. Overview

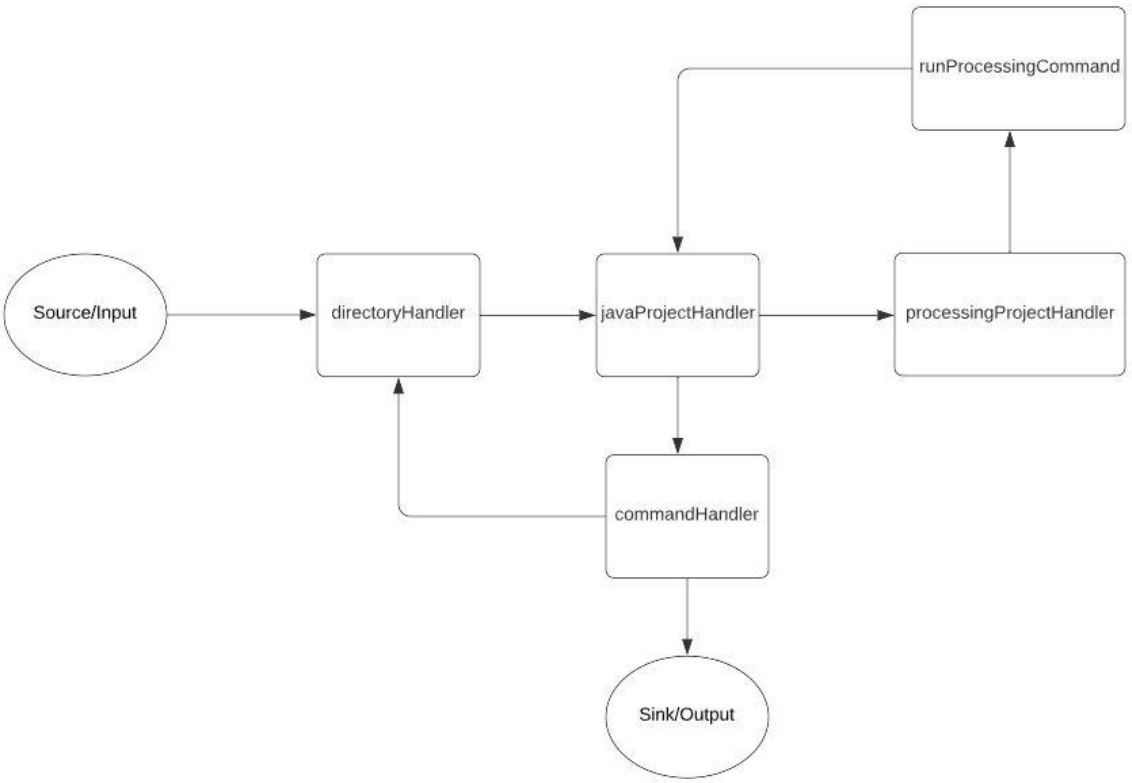
PuNIT is a framework for the automated testing of Processing programs. It allows for users to create custom tests for Processing projects.

The purpose of this document is to outline the architecture of the program , its intended use and a guide for use. As well as to propose future developments should the user desire to develop this program further.

2. Background

The purpose of this software application is to facilitate marking of student assignments for an introductory programming course easier and less time consuming. For example, if 800 students submit a Processing assignment and it takes a conservative estimate of five minutes to mark each assignment. That would take around 67 hours to complete the marking.

3. Architecture



The type of architecture used to design the program is a pipe and filter style. The application recursively goes through the submission folder and runs multiple tests on each of the submissions.

The input is a parent folder containing all of the student .pde submissions in individual folders named by student ID. First the directory handler function checks that a submission exists in the file. If the project is already in .java format then the file is passed to the commandHandler function which runs compile, static, and dynamic testing. If the student submission is still in .pde format, PuNIT will convert the file to a .java format and pass that file to the command handler function. Once each file has been tested the program will end and generate marks for each of the submissions and the results will be in a .csv file.

4. Types of Testing

Unit Testing: Individual methods and functions of the classes, components, or modules are tested.

Integration testing: These tests verify that different modules or services used by your application work well together

Performance Testing: These tests evaluate how a system performs under a particular workload. These tests help to measure the reliability, speed, scalability, and responsiveness of an application. For instance, a performance test can observe response times when executing a high number of requests, or determine how a system behaves with a significant amount of data. It can determine if an application meets performance requirements, locate bottlenecks, measure stability during peak traffic, and more.

Edge Case testing: To ensure the software was not behaving anomalously.

6. Runtime Protocol

PuNITs runtime depends on a few system requirements, which the user must address.

1. Java installed –

A java version of 16 or higher must be installed.

2. Installing processing –

The installation of processing can be found here: https://processing.org/download

PuNIT requires this due to the use of processing systems that allow for converting a processing project into a java project.

Whilst any version of processing should be alright as long as it contains processing-java, processing-4.0.1 is recommended.

3. Setting environment variables –

Once processing is installed, the user must add the processing directory to the PATH environment variable. The directory path should look like this for Windows:

\path\to\processing-#.#.#

And this for MacOS and Linux:

/path/to/processing-#.#.#

Windows

For Windows, navigate to the *Edit system environment variables* settings. Click on the *environment variables* button, where a list of user variables and system variables will then appear. Find the PATH variable in either of the user variables if you want only the current user to be able to use PuNIT or the system variable if you wish the whole system to be able to use PuNIT or both. Double-click on the PATH variable and click on the *new* button, where you will be prompted to add a new path, which will be the copied directory path to processing (\path\to\processing-#.#.#).

MacOS

For MacOS:

```
export PATH=$PATH:/path/to/processing-#.#.#
```

This will only be a temporary setting for the current shell session; if you want to set this PATH variable permanently, follow this tutorial: https://www.digitalocean.com/community/tutorials/how-to-view-and-update-the-linux-path-environment-variable

Linux

For Linux:

```
export PATH=$PATH:/path/to/processing-#.#.#
```

This will only be a temporary setting for the current shell session; if you want to set this PATH variable permanently, follow this tutorial:

https://www.digitalocean.com/community/tutorials/how-to-view-and-update-the-linux-path-environment-variable

6.1 Detailed Steps

PuNIT is automated where not much user input is required; initialising the testing process only requires a few user inputs.

U = User, S = System

1. S: requests directory, .pde or .java from U
2. U: inputs path to a directory, .pde or .java
    - \path\to\directory || /path/to/directory
    - \path\to*.pde || /path/to/*.pde
    - \path\to*.java || /path/to/*.java
3. S: requests .txt test file from U
4. U: inputs path to .txt test file
    - \path\to\TestFile.txt || /path/to/TestFile.txt
5. S: requests .xml static rules from U
6. U: inputs path to .xml static rules
    - \path\to\staticRules.xml || /path/to/staticRules.xml
7. S: Match to either:
    - Java project go to step 9
    - Processing project go to step 8
8. S: The processing project is converted:
    - Standard conversion
    - Conversion using --force means the folder existed already and had to be overridden

9. S: The project is tested for compilation

- CT passes

- CT fails

10. S: The project is tested for Static Testing

- ST passes

- ST fails

11. S: The project is tested for Dynamic Testing

-DT passes

-DT fails

12. S: Results for an individual student are generated into a directory found in the user's documents

- 20 marks for CT

- <= 5 marks for ST depending on testing

- Variable marks for DT depending on testing file flags

- If more assignments go to step 7

- If there are no more assignments, go to step 13

13. S: Test results of students are entered into an excel spreadsheet .xlsx

14. S: System exits

7. Input Options

The program can be run with the provided batch file for windows or shell script for mac and linux. The user will be prompted to enter the following file paths:

Input folder - the absolute path of the main directory of processing programs to be tested.
Test file - the absolute path of the dynamic TestFile.txt.
Static rules - the absolute path of the staticRules.xml file for PMD.

Alternatively these can be passed in a arguments if running the run scripts from the command line as follows:

run.(sh/bat) <Input Folder> <Test File> <Static Rules>

8. Output

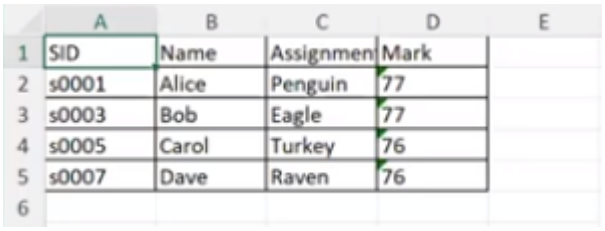
8.1. Text File Output

The program will generate a text file in the main directory called Results.txt. It will list the specific tests run and the results for each test ie. pass or fail, mark received. It will total the result at the bottom of the file.

```
Results for -
Student ID:  s0001 Name: Alice

################################################################################################################
################################################################################################################

COMPILE TEST

Compile Test Passed: 20

STATIC TEST

CyclomaticComplexity: ID: 1: 5
ExcessiveMethodLength: ID: 2: 5
TooManyFields: ID: 3: 5
TooManyMethods: ID: 4: 5
OneDeclarationPerLine: ID: 5: 4
FieldNamingConventions: ID: 6: 5
FormalParameterNamingConventions: ID: 7: 5
MethodNamingConventions: ID: 8: 5
ShortClassName: ID: 9: 5
ShortMethodName: ID: 10: 5
ShortVariable: ID: 11: 0

DYNAMIC TEST

Object Count: Penguin 7 = true: 3
Object Present: true: 5
################################################################################################################
################################################################################################################

Result Total = 77
```

8.2. Spreadsheet Output

The program will generate a spreadsheet in the main directory called studentsResults.xlsx. This contains each student's ID and name along with their calculated assignment mark. This can be viewed in any spreadsheet viewing/editing program.



	A	B	C	D	E
1	SID	Name	Assignmen	Mark	
2	s0001	Alice	Penguin	77	
3	s0003	Bob	Eagle	77	
4	s0005	Carol	Turkey	76	
5	s0007	Dave	Raven	76	
6					

8.3. Error Log File

If there is an exception thrown during the runtime of the program, an error log file will be generated in the root directory called log.txt. This includes your java version, operating system and stack trace of the exception. If you encounter and issue with the program, please submit an issue to the github repository with the log file and steps to replicate the issue.

```
Operating System: Linux
Java Version: 17.0.4

Exception: java.lang.RuntimeException: java.lang.ClassNotFoundException: MarchPenguin

sketchTest.<init>(sketchTest.java:37)
Punit.commandHandler(Punit.java:432)
Punit.processingProjectHandler(Punit.java:304)
Punit.RecursivePrint(Punit.java:213)
Punit.RecursivePrint(Punit.java:224)
Punit.RecursivePrint(Punit.java:224)
Punit.RecursivePrint(Punit.java:221)
Punit.RecursivePrint(Punit.java:221)
Punit.RecursivePrint(Punit.java:224)
Punit.RecursivePrint(Punit.java:224)
Punit.RecursivePrint(Punit.java:224)
Punit.main(Punit.java:133)
```

9. Commands

Two versions of the PuNIT system address the same problem with different takes. One version aims to provide the user with more of an interactive experience where they must utilise user commands to progress the runtime further. After discussing with the stakeholders, whilst the importance of user experience is detrimental, it was concluded that the user experience should be limited in favour of functionality. An essential user story was presented, which was the catalyst for this shift. Considering that a tutor, lecturer or academic could be marking assessments late at night, short on time, they would rather have a quick, automated experience that requires limited input.

Main_v3_user-input

User commands in this version include:

-ct – compile test

Performs the compile testing

-st – static test

Performs the static testing

-dt – dynamic test

Performs the dynamic testing

-o – output

Generates an output of completed testing

-e – exit

Exits the system

-h – help

It shows in the console a help message with commands and directions for more help

Main

This version of the system was reworked to be more automated, meaning that not much user input is required besides the program's initial arguments. The user can modify the test file to input tests for ST and DT and the static rules .xml to include or not include ST.

Test File

- Static Test example

-s 5 neededFiles: [Penguin.pde, Dog.pde]

- Dynamic Test example

-a 3 objectCount: [Penguin 7]

Static Rules

- Example

```
<rule ref="category/java/design.xml/TooManyFields">
  <properties>
    <property name="maxfields" value="15" />
  </properties>
</rule>
```

10. Post Developments

Further developments of this program can be done in several areas, namely in the expansion of the automated testing features. Since Processing programs have infinite variabilities in how they can be written, writing an automated testing suite has many challenges.

An approach that could foster easier automated testing would be to devise a strict set of guidelines for students writing processing projects, with the mechanics of testing in mind. Allowing for easier testing and less variation in student submissions. This may have other unwanted effects, such as making it harder to detect plagiarism or harder to teach independent thinking, if students are subject to strict guidelines. Therefore, a solution should be devised with beginner programmers and educators' input.
Another possible development is to host the command line application on university servers allowing students to upload their assignments via a file transfer protocol and then receive their results immediately.