Aristotle: A Flexible Open-Source Software Toolkit for Semi-Automated Marking of Programming Assignments

Michael D. Adams

Department of Electrical and Computer Engineering
University of Victoria
Victoria, BC, V8W 3P6, Canada

E-mail: `mdadams@ece.uvic.ca`

August 2017

Outline

Motivation

- many courses in engineering, computer science, and related disciplines require marking of programming assignments
- marking of programming assignments can be extremely time consuming
- marking typically entails:
  - check packaging
  - build and test code
  - may need to generate modified versions of student code
  - manual inspection of source code
  - prepare report to provide feedback to student about marking
- repeat for every student in class, which may be quite large
- tools that can automate or otherwise assist with some parts of assessment process highly beneficial
- many tools have been developed, but many also impose significant restrictions on user, such as requiring use of:
  - particular programming language (often Java)
  - particular software build tool or compiler tool chain
  - specific scripting language for customization or defining of test cases

Design Requirements

- tool to assist in semi-automated assessment of programming assignments
- accommodate wide range of assignment types
- allow any programming language to be used for assignments
- allow arbitrary tool to be used to build code in assignment submissions (for languages requiring compilation)
- provide direct support for use of build tools CMake and Make
- allow for wide range of testing methodologies
- decouple assignment submission from other operations and support variety of mechanisms for assignment submission, including:
  - GitHub Classroom
  - Git repositories
  - Zip and Gzipped-Tar archives
  - directories in local filesystem
- provide mechanism for creating modified version of student code
- allow automation to whatever extent is possible

Aristotle

- flexible open-source software toolkit for semi-automated marking of programming assignments
- automates most part of assessment process, including:
  - importing of assignment submissions
  - performing basic validity check on submissions
  - building and testing of code in submissions
  - generation of report summarizing results of building and testing code
- large degree of flexibility achieved by allowing behavior associated with various operations to be specified by user-provided programs (which can scripts using any scripting language)
- used earlier this year for teaching of fourth-year undergraduate and graduate courses with heavy programming content
- current version of Aristotle implemented using Bash scripts
- consists of approximately 8900 lines of code

Assignments and Workspaces

- **assignment submission** is arbitrary directory tree containing one or more files
- each assignment named by unique ID
- defined as collection of attribute-value pairs
- **attribute** is property of assignment (e.g., short assignment description and list of required files)
- all attribute-value pairs placed in what is called **assignment-definition file**
- for each assignment, user creates assignment-definition file
- assignment-definition files placed in user-configurable directory called **assignment-definition directory**
- many attributes have default values
- user need only specify value for particular attribute if has no default value or desired value differs from default
- assignment submission must be imported into assignment workspace
- **assignment workspace** is directory tree with particular layout known to Aristotle

Workflow

1. **import**: create new assignment workspace using assignment submission from some input source (e.g., Git repository or archive file)
2. **validate**: perform basic validity checks on data imported from assignment submission
3. **generate packages**: generate one or more variants of files from assignment submissions, where each variant is known as package
4. **configure**: perform any pre-build preparations for generated packages
5. **build**: build (e.g., compile and link) code for generated packages
6. **test**: test programs or libraries built for generated packages
7. **generate report**: generate report summarizing results of building and testing packages

Packages, Build Targets, and Test Targets

- each assignment has one or more packages
- **package** corresponds to transformed version of original assignment submission
- each package has number of build targets and test targets
- **build target** names artifact that can be generated by software build process
- **test target** names test that can be performed on software produced by build process
- packages, build targets, and test targets can be either
  - required (i.e., accessible to all users); or
  - optional (i.e., for instructor use only)

Palindrome Assignment Example

- student must write C++ program that reads whitespace-delimited words from standard input and writes to standard output whether each word is palindrome (i.e., program tests for palindromes)
- CMake to be used as build tool
- student must provide CMakeLists file for CMake that defines executable target named `is_palindrome`
- assignment to be named by assignment ID `palindrome`
- assignment submission has following three required files, all of which located in top-level directory of submission:
    1. `IDENTIFICATION.txt`, file which contains student and assignment information
    2. `CMakeLists.txt`, build configuration file for CMake
    3. `is_palindrome.cpp`, single C++ source code file for `is_palindrome` program
- complete example available as part of Aristotle software

Identification File `IDENTIFICATION.txt`

```
name "Jane Doe"
student_id "V12345678"
email "jdoe@uvic.ca"
section "T01"
assignment "palindrome"
```

Assignment-Definition File `palindrome.asgn`

```
# Specify a name or short description for the assignment.
name "Palindrome Test"

# Specify the files that must be included in the assignment submission.
required_files IDENTIFICATION.txt CMakeLists.txt is_palindrome.cpp

# Specify the files from the original submission that are to be included in
# the generated report.
report_files CMakeLists.txt is_palindrome.cpp

# Specify the packages defined for this assignment (i.e., one called
# "original").
packages original

########################################
# Information for "original" package.
########################################

# Specify a name or short description for the "original" package.
package-original/name "The original code exactly as submitted by the student."

# Specify the build targets for the "original" package (i.e., one called
# "is_palindrome").
package-original/builds is_palindrome
```

Assignment-Definition File `palindrome.asgn` (Continued)

```
# Specify the test targets for the "original" package (i.e., one called
# "is_palindrome").
package-original/tests is_palindrome

# Specify a name or short description for the "is_palindrome" build target.
package-original/build-is_palindrome/name "Build the is_palindrome program."

# Specify a name or short description for the "is_palindrome" test target.
package-original/test-is_palindrome/name "Test the is_palindrome program."

# Request that the test should only be performed if the associated build
# target "is_palindrome" is successfully built.
package-original/test-is_palindrome/depends_on build-is_palindrome

# Set the maximum time (in seconds) allowed for the test to 10.
package-original/test-is_palindrome/timeout 10

# Specify the command (including arguments) to be invoked to perform the test.
# Note: ${ARI_ASSIGNMENTS_DIR} is the directory containing this assignment
# definition file and ${ARI_DERIVED_DIR} is the directory containing the
# program to be tested.
package-original/test-is_palindrome/test \
  ${ARI_ASSIGNMENTS_DIR}/../private/bin/palindrome-is_palindrome-test
```

Test Script `palindrome-is_palindrome-test`

```
#! /usr/bin/env bash

is_palindrome="$ARI_DERIVED_DIR/is_palindrome"

failures=()
all=()

for word in even odd; do
    ari_cmpout -I "$word" -O "not palindrome\n" "$is_palindrome" || \
      failures+=("$word")
    all+=("$word")
done

for word in a aba abba; do
    ari_cmpout -I "$word" -O "palindrome\n" "$is_palindrome" || \
      failures+=("$word")
    all+=("$word")
done

exit_status=0
if [ ${#failures[@]} -gt 0 ]; then
    echo "${#failures[@]}/${#all[@]}" > "$ARI_TEST_RESULT_FILE"
    exit_status=1
fi

exit "$exit_status"
```

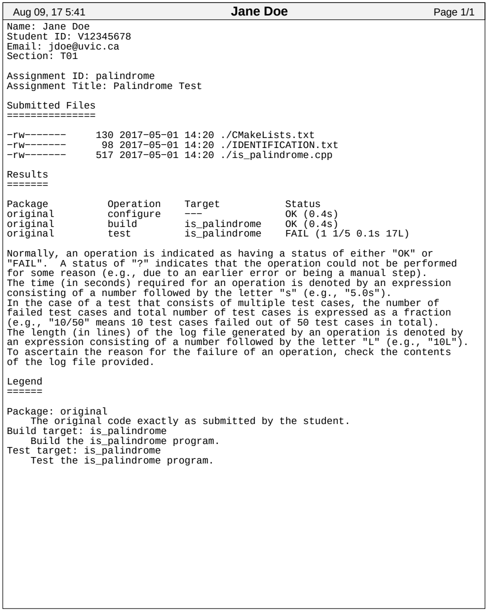
Command Sequence for Processing Assignment

- suppose that assignment submissions in Git repositories managed by GitHub Classroom using:
  - GitHub organization `$org`
  - assignment name `palindrome`
- can fully process all assignment submissions with following simple command sequence:

```
ari_import -p workspaces/ \
  $(ari_gc_lsrepo -o $org -f ssh -a palindrome)
ari_process workspaces/*
```

- for each submission, above command sequence will create assignment workspace under directory `workspaces` with each workspace containing generated report in file named `report.pdf`

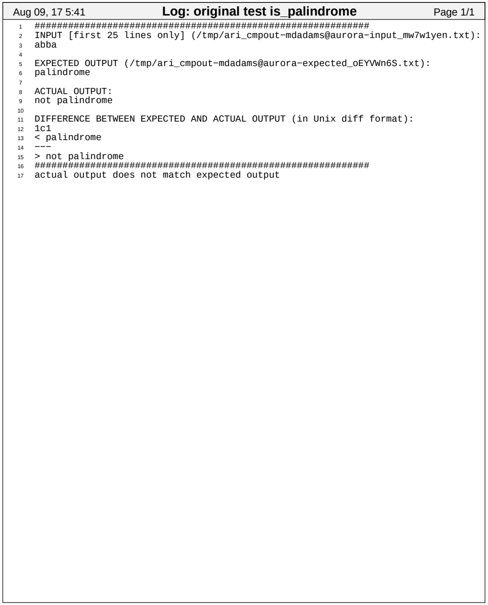
Report Summary Page Example



Aug 09, 17 5:41 **Jane Doe** Page 1/1

```
Name: Jane Doe
Student ID: V12345678
Email: jdoe@uvic.ca
Section: T01

Assignment ID: palindrome
Assignment Title: Palindrome Test

Submitted Files
===============

-rw-------     130 2017-05-01 14:20 ./CMakeLists.txt
-rw-------      98 2017-05-01 14:20 ./IDENTIFICATION.txt
-rw-------     517 2017-05-01 14:20 ./is_palindrome.cpp

Results
=======

Package          Operation    Target           Status
original         configure    ---              OK (0.4s)
original         build        is_palindrome    OK (0.4s)
original         test         is_palindrome    FAIL (1 1/5 0.1s 17L)

Normally, an operation is indicated as having a status of either "OK" or
"FAIL".  A status of "?" indicates that the operation could not be performed
for some reason (e.g., due to an earlier error or being a manual step).
The time (in seconds) required for an operation is denoted by an expression
consisting of a number followed by the letter "s" (e.g., "5.0s").
In the case of a test that consists of multiple test cases, the number of
failed test cases and total number of test cases is expressed as a fraction
(e.g., "10/50" means 10 test cases failed out of 50 test cases in total).
The length (in lines) of the log file generated by an operation is denoted by
an expression consisting of a number followed by the letter "L" (e.g., "10L").
To ascertain the reason for the failure of an operation, check the contents
of the log file provided.

Legend
======

Package: original
    The original code exactly as submitted by the student.
Build target: is_palindrome
    Build the is_palindrome program.
Test target: is_palindrome
    Test the is_palindrome program.
```

Report Error Log Example



Aug 09, 17 5:41 **Log: original test is_palindrome** Page 1/1

```
################################################################
INPUT [first 25 lines only] (/tmp/ari_cmpout-mdadams@aurora-input_mw7w1yen.txt):
abba

EXPECTED OUTPUT (/tmp/ari_cmpout-mdadams@aurora-expected_oEYVWn6S.txt):
palindrome

ACTUAL OUTPUT:
not palindrome

DIFFERENCE BETWEEN EXPECTED AND ACTUAL OUTPUT (in Unix diff format):
1c1
< palindrome
---
> not palindrome
################################################################
actual output does not match expected output
```

Conclusions

- developed new flexible open-source software toolkit for semi-automated assessment of programming assignments
- toolkit can be used with any programming language, build tools, and compiler tool chains
- by allowing user to customize various key operations in processing of assignments, great flexibility is achieved
- software can be obtained from official Git repository on GitHub:
  - `https://github.com/mdadams/aristotle`
- by using software like Aristotle, amount of work required to mark programming assignments can be greatly reduced

Questions?