Version Control Standards:

For version control, we use Github.

Checking out can be done by anyone. Typically, one checks out a module they were assigned to work on. If the module is incomplete or has problems, a branch is created for said module so it may be worked on and fixed.

Tag naming convention is to start with the name of the module and include a .1 after the name. We increase by .1 each time the module is worked on until it is finished. The final module is pushed to the master branch with the original name and no decimals after. Final push may only be done once the code has been peer-reviewed and all unit tests return no errors.

Tests are run each time a change is made to a module. Full system tests are run once all module are complete.

Unit Testing Standards:

For unit testing, we use the CUNIT testing framework as taught in class. Each module is tested as it is worked on, however, some basic modules may not require unit testing (i.e. getting user input and other simple tasks).

Below is an example of a test makefile for the calc_roots module where test1 and test2 refer to the unit tests that a person has written:

:

```
CC = gcc
CFLAGS = -std=c99 -Wall -Werror -pedantic
LDFLAGS = -lm

calc_roots.o: calc_roots.c

test1: test1.c ../cunit/cunit.o ../num_roots/num_roots.o ../sqrt/sqrt.o calc_roots.o

test2: test2.c ../cunit/cunit.o ../num_roots/num_roots.o ../sqrt/mock_sqrt.o calc_roots.o

test: test1 test2
	./test1
	./test2

.PHONY: clean
clean:
	-rm calc_roots.o test1 test2
```

All test makefiles are done using this structure. As for the tests themselves, it is up to the person designated to work on the module to create them at their discretion. The tests should follow the guidelines that are discussed in our XP Textbook.

Complete Functional Test Standards:

To run a complete functional test on the quad solver program, a makefile has been created in the Build directory that compiles all the code files and runs all tests. Simply go to the Build directory and run make followed by make test.

In addition, all unit tests and integration tests can be run from the Deployment directory. The makefile in Deployment allows for all tests to be run by using the command make followed by make test, or a user can run make unit-test to only run the unit tests or make integration -test to run only the integration tests.

Automation Standard:

We used a standard makefile to automate the building and running of the program.

Programming Standards:

The chosen coding standard we used is located here: https://users.ece.cmu.edu/~eno/coding/CCodingStandard.html

To summarize, each code file must have the name of the file, date, and a brief description at the top of the file like so:

```
/**
 * @file calc_roots.c
 * @date Feb 15, 2017
 * @brief Calculate quadratic roots.
 */
```

Next is all the include files necessary for the code including header files which each code file should have.

Below that, there is a longer description of what the code does as well as the parameters used and what the output is. For example, here is the calc_roots code file description with the parameters:

```
/**
 * @brief Calculates the roots for the given quadratic equation puts the
 * results in the root structure.
 *
 * @param coef Coefficients to caluclate the roots of.
 * @param num_roots Number of real roots.
 * @param root Out parameter to write the values of the roots into.
 *
 * @return True on success, false on error.
 */
```

Additionally, each code file should have logging so that the code output may be easily inspected by other team members.

IDE Standards:

Programmers may use any text editor that they are comfortable with. Sublime is recommended. Compiling of code is done with gcc.

Makefile Standards:

Makefiles are to be used for compiling a code module, running all functional and unit tests, and packaging up the entire program. Syntax and commands used are found in the example below:

```
1  CC = gcc
2  CFLAGS = -std=c99 -Wall -Werror -pedantic
3
4  input.o: input.c
5
6  .PHONY: clean
7  clean:
8          -rm input.o
```

This is the common setup of a Makefile for a module that does not require tests. (Such as taking in user input). If a module has tests, the Makefile will look like this:

```
CC = gcc
CFLAGS = -std=c99 -Wall -Werror -pedantic

input_val.o: input_val.c

test1: test1.c ../cunit/cunit.o input_val.o

test2: test2.c ../cunit/cunit.o input_val.o

.PHONY: test
test: test1 test2
        ./test1
        ./test2

.PHONY: clean
clean:
        -rm input_val.o test1 test2
```

License:

GNU General Public License