NEOVIM AUTOMATED TESTING FRAMEWORK

Brandi Durham, Lauren Tubbs, Misae Evans, John Quinn

CSCI 362 TERM PROJECT

Neovim Automated Testing Framework

Chapter One: Building Neovim

Building Neovim from source was a simple task. The instructions provided by the Neovim team were outstanding. To build Neovim we used Ubuntu 16.04.

First, we got the project prerequisites with the following command:

```
sudo apt-get install ninja-build gettext libtool libtool-bin autoconf automake cmake g++ pkg-config unzip
```

Then, we cloned the repository with:

```
git clone https://github.com/neovim/neovim
```

Then, we built the source code with:

```
cd neovim
make
sudo make install
```

To open Neovim simply type "nvim". If you wish to exit Neovim you must use the :q command. To run the existing tests:

```
cd ~/neovim
make test
```

Running the tests takes approximately ten minutes, 636 tests were run and all passed.

Chapter 2: Test Plan

Testing Process

Our framework will add to the functional tests already in place for testing the functionality of the neovim commands.

Requirements Traceability

Commands are key features in neovim's function, and if they operate as intended then the requirements are met.

TESTED ITEMS

Neovim commands

TESTING SCHEDULE

- October 15th - November 9th (3rd deliverable): develop test code; Automated testing framework
- November 9th - November 19th (4th deliverable): Create 25 test cases; run automated testing
- November 19th-November 25th (5th deliverable): Design and inject 5 faults; exercise framework and analyze results

TEST RECORDING PROCEDURES

Redirect test results to text file.

HARDWARE AND SOFTWARE REQUIREMENTS

Linux system required.

CONSTRAINTS

There are a mere four of us in a group for a large number of issues, we may not be able to touch on the more detailed issues.

CHAPTER THREE: AUTOMATED TESTING FRAMEWORK

Our framework is set up so that a tester simply has to add two files in the appropriate folders to add a new test:

- A text file to TestAutomation/testCasesExecutables which contains a sequence of Neovim commands. Name it test01.txt, with the 01 changed to the appropriate number.
- A text file to TestAutomation/testCases containing the desired result of executing those Neovim commands. Name it correct-test01.txt, with the 01 changed to the appropriate number. Any non-alphanumeric characters need to be surrounded by angle brackets and called by the name used by PyAutoGUI (see here). For example, a spacebar input is written <space>and the escape key is <esc>.

After adding all desired tests, run the script TestAutomation/scripts/runAllTests.sh. An HTML report will be generated in TestAutomation/reports.

Description of the Framework

This bash script iterates through all the files in testCasesExecutables. For each file:

- The bash script calls the Python script TestAutomation/scripts/testInterpreter.py with appropriate arguments.
- The Python script parses the text file to extract the Neovim commands and uses the module PyAutoGUI to simulate keyboard input to Neovim. (Almost all of the test cases create a text file which is stored in TestAutomation/temp.)
- The bash script compares the file generated by Neovim commands to the correct file in TestAutomation/testCases using a "diff". If they are the same, the test is passed. After running all the tests, the bash script generates an HTML report of the test results and puts it in TestAutomation/reports.

Test Case Example

The tester is assumed to have no programming or technical knowledge, thus we have created a sort of language for the tester to use when creating test cases. Take the following test:

Test ability to write and save a new file (:wq command). In order to perform this test, the tester would save the text: nvim <space> test01-output.txt <enter> <i> Testing the insert function <enter> <esc> :wq <enter> into a text file. The bash script would run a python script to translate the text in the text file to commands via the pyautogui library and run the test.

Chapter Four: 25 Test Cases

System tests

1. Test ability to write and save a new file (:wq command).
2. Test ability to append to an existing file (:o command).
3. Test ability to open a file, enter text, and close without saving (:q! command).
4. Test ability to delete a character (x command).
5. Test ability to undo changes to the last line (u command).
6. Test ability to move cursor to beginning of file (gg command).
7. Test ability to move cursor to end of line ($ command).
8. Test ability to move cursor to end of a word (e command).
9. Test ability to move cursor to beginning of a word (b command).
10. Test ability to move cursor to end of file (G command).
11. Test ability to find the first instance of a pattern in text (/ command).

12. Test ability to find the first instance of a pattern in text (2/ command).
13. Test ability to find the last instance of a pattern in text (? command).
14. Test ability to find the second-to-last instance of a pattern in text (2? command).
15. Test ability to move through found instances of a pattern in text (n command).
16. Test another way to have the ability to delete character (X command).
17. Test ability to delete an entire line (D command).
18. Test the ability to undo text and then redo text (combination of u command and ctrl-R command).
19. Test the ability to move up in neovim text and delete text (combination k command and D command).
20. Test the ability to move down in neovim text and delete text(combination of j, k, D command).
21. Test another way to move down in neovim text and delete text(combination of + and D command).
22. Test another way to move up in neovim text and delete text (combination of - and D command).
23. Test the ability to replace a char with another char (r command).
24. Test replace mode for text insertion (R command).
25. Test the repeat last change function (. command).

Chapter Five: Fault Injection

We have injected five faults into the neovim source code and analyzed the results. Documented below where the faults are and what was changed:

1. cause the mouse to not response correctly.

   ```
   ~/Desktop/neovim/src/nvim/tui/input.c
   if (button != 0 || (ev != TERMKEY_MOUSE_PRESS && ev != TERMKEY_MOUSE_DRAG
   && ev != TERMKEY_MOUSE_RELEASE)) {
   return;
   }
   ```
   changed '==0' to '!=0

2. make the coordinates of where the cursor is go up instead of down, or the opposite of what it's suppose to do

   ```
   ~/Desktop/neovim/src/nvim/tui/input.c
   row**++**; col--; // Termkey uses 1-based coordinates
   ```
   changed -- to ++

3. cause the buttons to choose the wrong direction. I switched the 1 and 2, so it will go the opposite directions than planned.

```
~/Desktop/neovim/src/nvim/tui/input.c
if (button == 2) {
len += (size_t)snprintf(buf + len, sizeof(buf) - len, "Left");
} else if (button == 1) {
len += (size_t)snprintf(buf + len, sizeof(buf) - len, "Middle");
} else if (button == 3) {
len += (size_t)snprintf(buf + len, sizeof(buf) - len, "Right");
}
```

Switched 1 and 2

4. Should go for an extra loop, means the startup will be longer by 1ms.

```
~/Desktop/neovim/src/nvim/tui/tui.c

for (size_t ms = 0; ms <= 100 && !tui_is_stopped(ui);) {
ms += (loop_poll_events(&tui_loop, 20) ? 20 : 1);
}
```

Changed < to <=

5. Switched the HL_ITALIC and HL_BOLD

```
~/Desktop/neovim/src/nvim/tui/tui.c

int attr = ui->rgb ? attrs.rgb_ae_attr : attrs.cterm_ae_attr;
bool bold = attr & HL_ITALIC;
bool italic = attr & HL_BOLD;
```

The expected results of the test cases with the fault injections implemented were that the test cases would fail. However, even with the faults injected the test cases passed.