Usability Comparison of Vim and Nano Code Editors

A Technical White Paper

```
                 :::
  iLE88Dj.  :jD88888Dj:
.LGitE888D.f8GjjjL8888E;
iE   :8888Et.      .G8888.
;i    E888,         ,8888,
      D888,         :8888:
      D888,         :8888:
      D888,         :8888:
      D888,         :8888:
      888W,         :8888:
      W88W,         :8888:
      W88W:         :8888:
      DGGD:         :8888:
                    :8888:
                    :W888:
                    :8888:
                     E888i
                     tW88D
```

Prepared by: Thomas Schmidt

March 27, 2025

Contents

Introduction

Terminal-based text editors like Vim and Nano are indispensable for developers, system administrators, and IT professionals who need lightweight, efficient tools for editing code and configuration files directly in a command-line environment. This white paper presents a cognitive walkthrough usability test comparing Vim and Nano, focusing on their effectiveness for intermediate users—those with basic terminal experience but minimal prior knowledge of these editors. These users, typically programmers or sysadmins, value tools that are efficient, intuitive, and provide clear feedback for tasks like coding, searching, and file management. This study evaluates how well each editor supports such users in completing ten common editing tasks, aiming to identify which editor better meets their needs and offer actionable recommendations for improvement.

Methodology

Test Environment

The test was conducted on Windows 11, using Windows Terminal with SSH to a Linux server (Ubuntu 22.04). Vim (version 9.1) and Nano (version 7.2) were installed via `apt`. Testing occurred in a distraction-free home workspace.

Tasks

Ten tasks were designed to reflect typical editing activities:

1. Open a new file named ”test.txt”.
2. Write a sample Python function.
3. Save the file.
4. Search for a specific word in the file.
5. Replace all instances of ”print” with ”echo”.
6. Undo the last change.
7. Navigate to line 5 of the file.
8. Delete line 5.
9. Copy and paste a line of text.
10. Exit the editor.

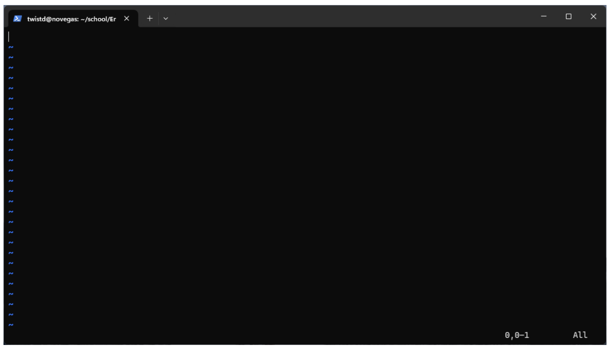
Evaluative Criteria

Each task was scored on a five-point Likert scale (1 = Strongly Disagree, 5 = Strongly Agree) across five metrics: Efficiency, Intuitiveness, Feedback, Error Tolerance, and Engagement.

Results

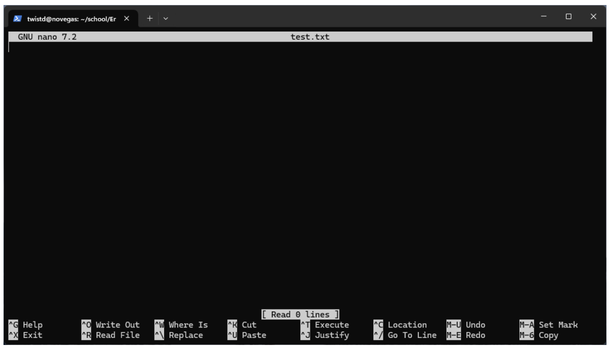
Task 1: Open a New File Named ”test.txt”

In Vim, typing `vim test.txt` opened the editor in command mode, showing a blank screen with no immediate editing cues. Nano’s `nano test.txt` displayed a clean interface with command hints at the bottom, making it more approachable.



```
~
~
~
                                                  0,0-1         All
```

Figure 1: Vim opening ”test.txt” in command mode (Figure 1).



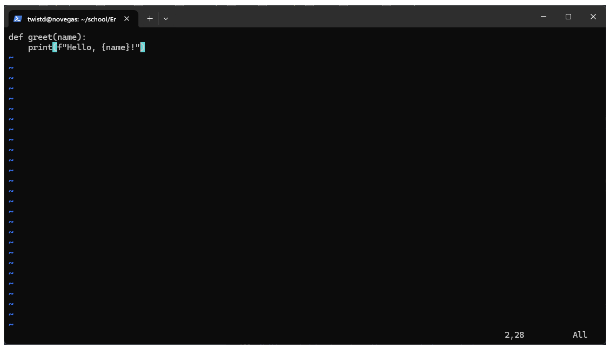
```
  GNU nano 7.2                          test.txt

                              [ Read 0 lines ]
^G Help       ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo      M-A Set Mark
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line M-E Redo      M-6 Copy
```

Figure 2: Nano opening ”test.txt” with command hints (Figure 2).

Scores: Vim: 15 (Eff: 3, Int: 3, Feed: 4, Err: 4, Eng: 3); Nano: 21 (Eff: 4, Int: 5, Feed: 4, Err: 4, Eng: 4).

Task 2: Write a Sample Python Function

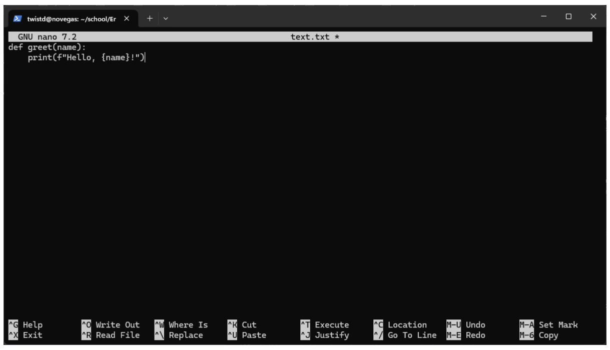
Vim required pressing `i` to enter insert mode before typing `def greet(): print("Hello")`. Without prior knowledge, this mode switch was confusing. Nano allowed immediate typing, enhancing usability for beginners.



twistd@novegas: ~/school/Er

```
def greet(name):
    print(f"Hello, {name}!")
~
~
~
                                                         2,28          All
```

Figure 3: Vim in insert mode with a Python function (Figure 3).



twistd@novegas: ~/school/Er

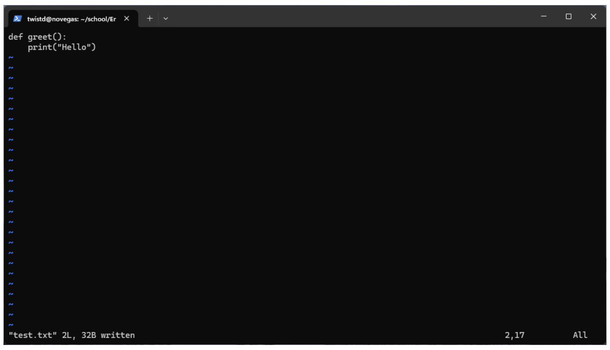
```
  GNU nano 7.2                              text.txt *
def greet(name):
    print(f"Hello, {name}!")




^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location  M-U Undo     M-A Set Mark
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^/ Go To Line M-E Redo     M-6 Copy
```

Figure 4: Nano with the same Python function entered (Figure 4).

Scores: Vim: 16 (Eff: 4, Int: 2, Feed: 3, Err: 4, Eng: 3); Nano: 25 (Eff: 5, Int: 5, Feed: 5, Err: 5, Eng: 5).

Task 3: Save the File

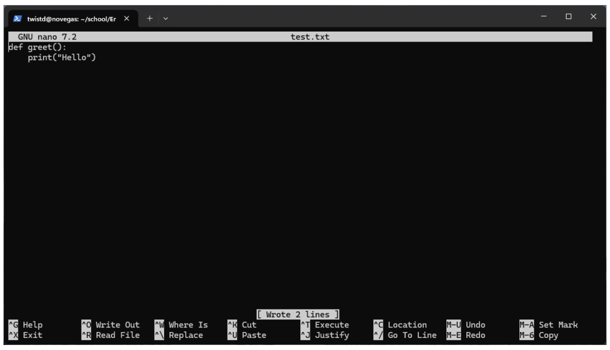
Vim's `:w` command saved the file, but its syntax was unintuitive. Nano's `Ctrl+O` prompted a save with clear instructions, improving discoverability.



twistd@novegas: ~/school/Er

```
def greet():
    print("Hello")
~
~
~
"test.txt" 2L, 32B written                               2,17          All
```

Figure 5: Vim saving "test.txt" with :w command (Figure 5).



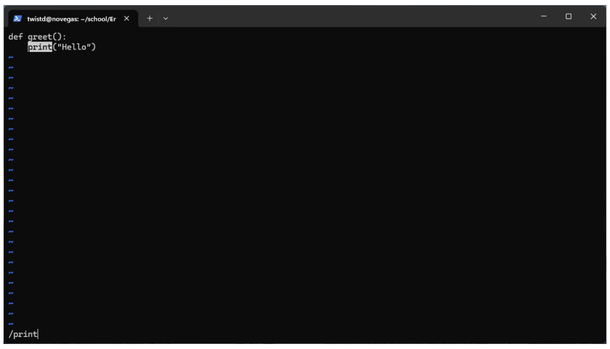
```
  GNU nano 7.2                         test.txt
def greet():
    print("Hello")




                               [ Wrote 2 lines ]
^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location  M-U Undo     M-A Set Mark
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^/ Go To Line M-E Redo     M-6 Copy
```

Figure 6: Nano save prompt with Ctrl+O (Figure 6).

Scores: Vim: 17 (Eff: 4, Int: 3, Feed: 3, Err: 4, Eng: 3); Nano: 22 (Eff: 4, Int: 5, Feed: 4, Err: 4, Eng: 5).

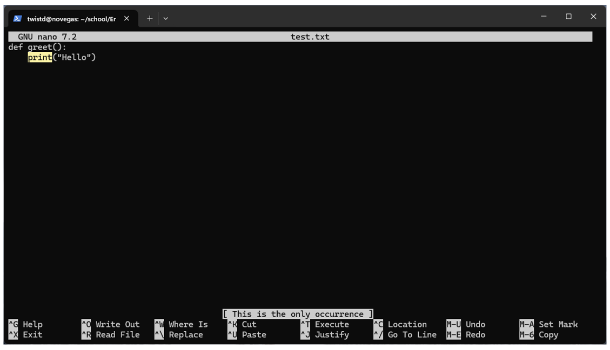
Task 4: Search for a Specific Word

Vim's `/print` searched effectively, highlighting matches, but required prior knowledge. Nano's `Ctrl+W` opened a visible search prompt, aiding novices.



```
def greet():
    print("Hello")
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
/print
```

Figure 7: Vim searching for "print" (Figure 7).



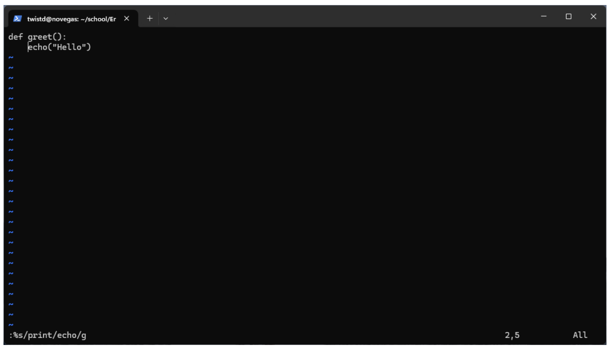
```
  GNU nano 7.2                         test.txt
def greet():
    print("Hello")




                          [ This is the only occurrence ]
^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location  M-U Undo     M-A Set Mark
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^/ Go To Line M-E Redo     M-6 Copy
```

Figure 8: Nano's search prompt for "print" (Figure 8).

Scores: Vim: 19 (Eff: 4, Int: 3, Feed: 5, Err: 4, Eng: 3); Nano: 20 (Eff: 4, Int: 4, Feed: 4, Err: 4, Eng: 4).

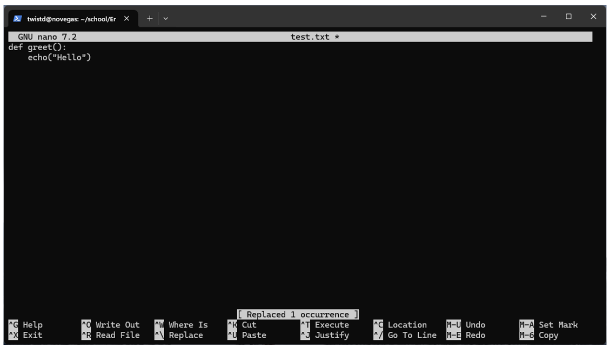
Task 5: Replace All Instances of ”print” with ”echo”

Vim’s `:%s/print/echo/g` efficiently replaced all instances, though the command was cryptic. Nano’s `Ctrl+\` required manual confirmation per instance, lacking a global option.



```
def greet():
    echo("Hello")
:%s/print/echo/g                                              2,5           All
```

Figure 9: Vim replacing ”print” with ”echo” (Figure 9).



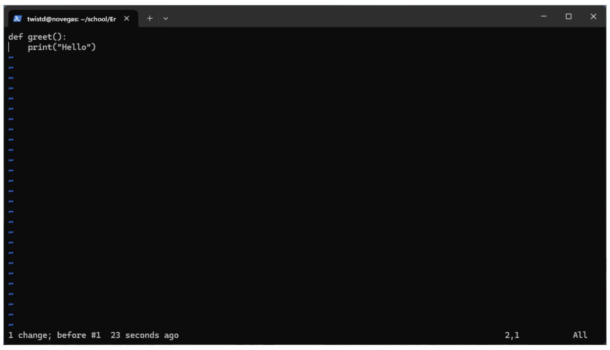
```
  GNU nano 7.2                          test.txt *
def greet():
    echo("Hello")

                               [ Replaced 1 occurrence ]
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo      M-A Set Mark
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line M-E Redo      M-6 Copy
```

Figure 10: Nano’s replace prompt in action (Figure 10).

Scores: Vim: 20 (Eff: 5, Int: 3, Feed: 4, Err: 4, Eng: 4); Nano: 15 (Eff: 2, Int: 4, Feed: 3, Err: 3, Eng: 3).

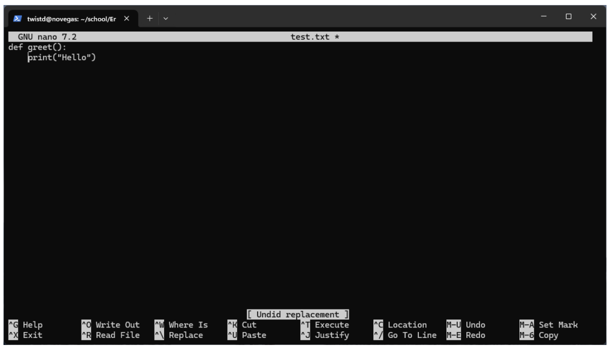
Task 6: Undo the Last Change

Vim’s `u` instantly reverted the replace, with clear feedback. Nano’s `Alt+U` worked but was less obvious without hints.



```
def greet():
    print("Hello")
1 change; before #1  23 seconds ago                           2,1           All
```

Figure 11: Vim undoing the replace operation (Figure 11).



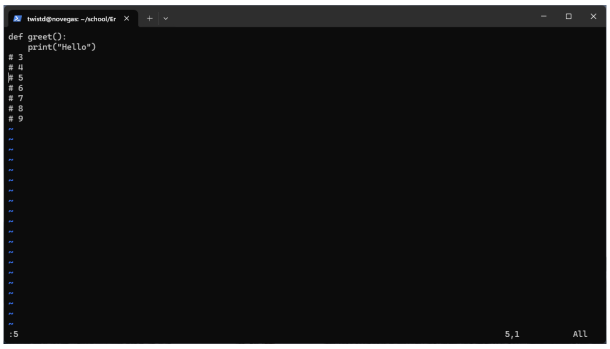
```
twistd@novegas: ~/school/Er
  GNU nano 7.2                          test.txt *
def greet():
    print("Hello")





                              [ Undid replacement ]
^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location   M-U Undo     M-A Set Mark
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^/ Go To Line M-E Redo     M-6 Copy
```

Figure 12: Nano after undoing the replace (Figure 12).

Scores: Vim: 22 (Eff: 5, Int: 4, Feed: 5, Err: 4, Eng: 4); Nano: 18 (Eff: 4, Int: 3, Feed: 4, Err: 3, Eng: 4).

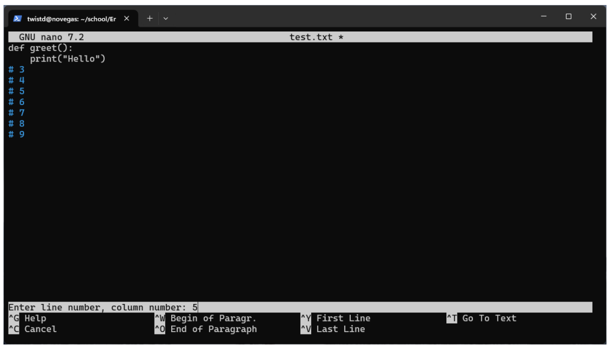
Task 7: Navigate to Line 5

Vim's `:5` moved to line 5 quickly, but required command familiarity. Nano's `Ctrl+_` prompted for a line number, guiding users.



```
twistd@novegas: ~/school/Er
def greet():
    print("Hello")
# 3
# 4
# 5
# 6
# 7
# 8
# 9
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
:5                                                         5,1           All
```

Figure 13: Vim navigating to line 5 (Figure 13).



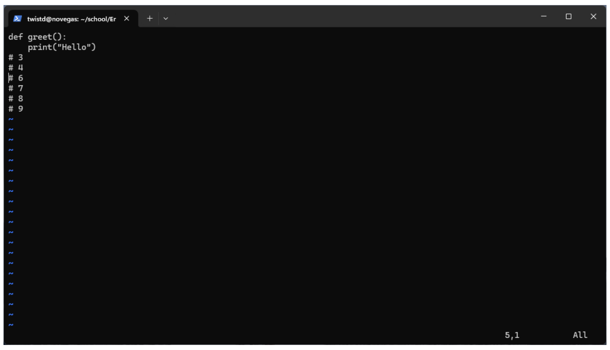
```
twistd@novegas: ~/school/Er
  GNU nano 7.2                          test.txt *
def greet():
    print("Hello")
# 3
# 4
# 5
# 6
# 7
# 8
# 9




Enter line number, column number: 5
^G Help              ^W Begin of Paragr.    ^Y First Line        ^T Go To Text
^C Cancel            ^O End of Paragraph    ^V Last Line
```

Figure 14: Nano's line navigation prompt (Figure 14).

Scores: Vim: 19 (Eff: 5, Int: 3, Feed: 4, Err: 4, Eng: 3); Nano: 20 (Eff: 4, Int: 4, Feed: 4, Err: 4, Eng: 4).

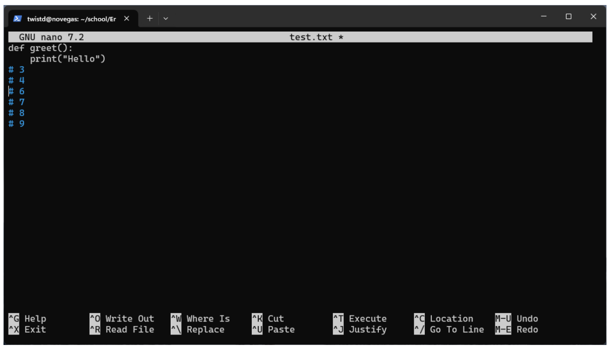
Task 8: Delete Line 5

Vim's `dd` deleted line 5 efficiently after navigation. Nano's `Ctrl+K` cut the line with visible cursor feedback.



```
twistd@novegas: ~/school/Er
def greet():
    print("Hello")
# 3
# 4
# 6
# 7
# 8
# 9
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
                                                                5,1           All
```

Figure 15: Vim after deleting line 5 (Figure 15).



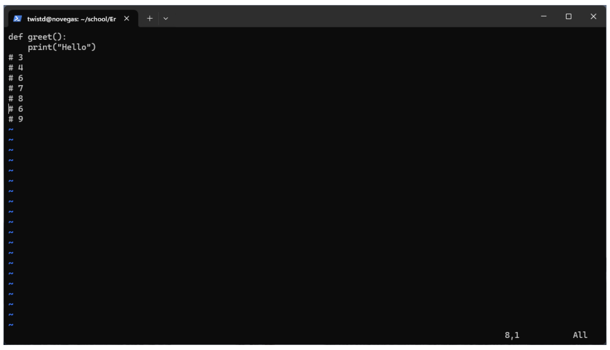
```
twistd@novegas: ~/school/Er
  GNU nano 7.2                           test.txt *
def greet():
    print("Hello")
# 3
# 4
# 6
# 7
# 8
# 9









^G Help       ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line M-E Redo
```

Figure 16: Nano after cutting line 5 (Figure 16).

Scores: Vim: 20 (Eff: 5, Int: 3, Feed: 4, Err: 4, Eng: 4); Nano: 21 (Eff: 4, Int: 4, Feed: 5, Err: 4, Eng: 4).

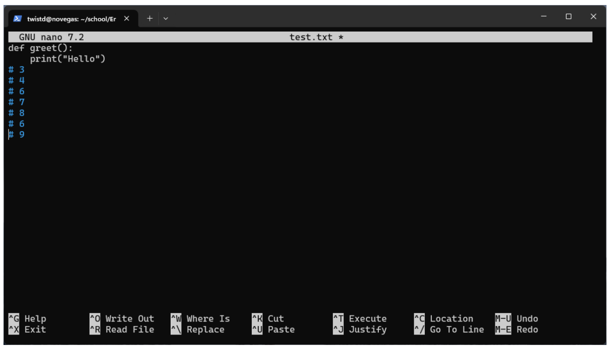
Task 9: Copy and Paste a Line of Text

Vim's `yy` and `p` copied and pasted a line quickly, but were unintuitive. Nano's `Alt+6` and `Ctrl+U` were slower but better documented.



```
twistd@novegas: ~/school/Er
def greet():
    print("Hello")
# 3
# 4
# 6
# 7
# 8
# 6
# 9
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
                                                                8,1           All
```

Figure 17: Vim after pasting a copied line (Figure 17).



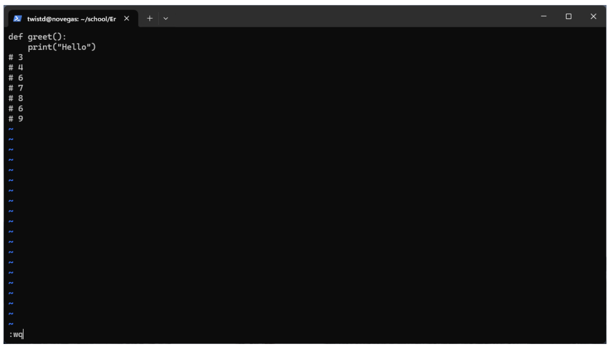
```
twistd@novegas: ~/school/Er
  GNU nano 7.2                                   test.txt *
def greet():
    print("Hello")
# 3
# 4
# 6
# 7
# 8
# 6
# 9

^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line M-E Redo
```

Figure 18: Nano after pasting a copied line (Figure 18).

Scores: Vim: 18 (Eff: 4, Int: 3, Feed: 4, Err: 4, Eng: 3); Nano: 19 (Eff: 3, Int: 4, Feed: 4, Err: 4, Eng: 4).

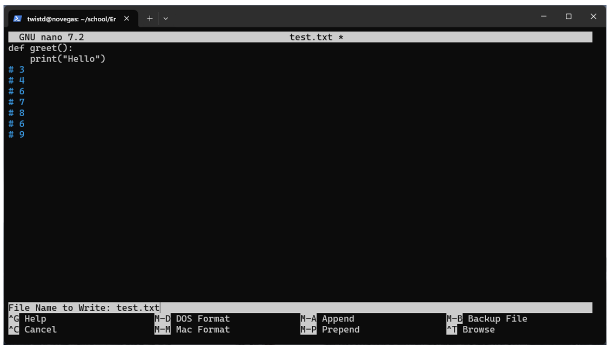
Task 10: Exit the Editor

Vim's `:q` (or `:wq`) exited, but was tricky without syntax knowledge. Nano's `Ctrl+X` with a save prompt was straightforward.



```
twistd@novegas: ~/school/Er
def greet():
    print("Hello")
# 3
# 4
# 6
# 7
# 8
# 6
# 9
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
:wq
```

Figure 19: Vim with :q command entered (Figure 19).



```
twistd@novegas: ~/school/Er
  GNU nano 7.2                                   test.txt *
def greet():
    print("Hello")
# 3
# 4
# 6
# 7
# 8
# 6
# 9

File Name to Write: test.txt
^G Help              M-D DOS Format       M-A Append           M-B Backup File
^C Cancel            M-M Mac Format       M-P Prepend          ^T Browse
```

Figure 20: Nano's exit prompt with Ctrl+X (Figure 20).

Scores: Vim: 16 (Eff: 3, Int: 3, Feed: 3, Err: 4, Eng: 3); Nano: 22 (Eff: 4, Int: 5, Feed: 4, Err: 4, Eng: 5).

Cumulative Scores

Criteria	**Vim**	**Nano**
Efficiency	42	38
Intuitiveness	30	42
Feedback	39	41
Error Tolerance	40	39
Engagement	33	41
Total	184	201

Table 1: Cumulative usability scores for Vim and Nano.

Conclusion

Nano outperformed Vim (201 vs. 184), excelling in intuitiveness and engagement due to its clear interface and on-screen hints. Vim showed higher efficiency for advanced tasks (e.g., Task 5), reflecting its power-user focus, but its learning curve hindered novices. Nano suits quick, accessible editing, while Vim rewards mastery with speed and control.

Recommendations

- **Vim**: Add a beginner mode with command hints.
- **Nano**: Implement global replace functionality.
- **Both**: Enhance accessibility with screen reader support.

Bibliography

References

[1] Vim. (2025). Home. `https://www.vim.org/`

[2] Nano. (2025). Home. `https://www.nano-editor.org/`

Appendix

Task	Editor	Eff.	Int.	Feed.	Err.	Eng.
1	Vim	3	3	4	4	3
1	Nano	4	5	4	4	4
2	Vim	4	2	3	4	3
2	Nano	5	5	5	5	5
3	Vim	4	3	3	4	3
3	Nano	4	5	4	4	5
4	Vim	4	3	5	4	3
4	Nano	4	4	4	4	4
5	Vim	5	3	4	4	4
5	Nano	2	4	3	3	3
6	Vim	5	4	5	4	4
6	Nano	4	3	4	3	4
7	Vim	5	3	4	4	3
7	Nano	4	4	4	4	4
8	Vim	5	3	4	4	4
8	Nano	4	4	5	4	4
9	Vim	4	3	4	4	3
9	Nano	3	4	4	4	4
10	Vim	3	3	3	4	3
10	Nano	4	5	4	4	5

Table 2: Raw usability scores for all tasks.