Experimental Comparison of the Effectiveness and Error Rates of Projectional and Text Editors

Tomáš Petro
Department of Computers and Informatics, Technical University of Košice, Slovakia

Jaroslav Porubän[1]
Department of Computers and Informatics, Technical University of Košice, Slovakia

Abstract

Editors can significantly influence user experience, as well as factors such as programming speed and the occurrence of syntactic errors. Given our experience in programming in the projectional editor MPS, which we believe offers numerous advantages, we decided to conduct an experiment involving students with limited programming experience. The aim was to determine if a novice programmer could program in projectional editor more effectively than in traditional text editor.

A total of 83 first-year computer science students from our university participated in the experiment. The methodology encompassed the selection of an appropriate programming language for the experiment, customization of the language's editor, preliminary trials, the main experiment, and concluding evaluations of speed and error rates associated with both editors. Additionally, the study sought insights into whether students exhibited a preference for programming in a projectional editor or adhering to a classic text editor paradigm.

2012 ACM Subject Classification Software and its engineering → Integrated and visual development environments

Keywords and phrases Text editor, Projectional editor, Effectiveness, Experiment

Digital Object Identifier 10.4230/OASIcs.SLATE.2024.5

Funding This work was supported by project VEGA No. 1/0630/22 Lowering Programmers Cognitive Load Using Context-Dependent Dialogs.

1 Introduction

Projectional editors also known as structure editors are formed around direct editing of abstract document's structure, not directly the text manipulation. While the concept is not the new one [3], projectional editors are relatively unfamiliar to most programmers. In the last years, a lot of effort was invested into popularising and improving projectional editors [5, 6, 7]. From an educational standpoint, individuals typically learn to work with text editors such as Vim, Notepad, or programming-specific ones like VS Code, CLion, IntelliJ, and others. However, despite this prevalent familiarity with text-based editors, one company has successfully developed a projectional editor that is actively utilized in various projects within companies. This editor, named MPS by JetBrains, prompted our interest in comparing it with a traditional text editor in terms of coding speed and error occurrence.

One rationale for conducting such a comparison lies in the manifold enhancements offered by projectional editors over their text-based counterparts. These enhancements include syntax checking during code composition, clear formatting of written code (such as automatic bracket completion, a feature specific to each language editor within MPS), among others. A similar test was conducted on a small sample of programmers, many of whom were university graduates with over five years of programming experience in various companies [1]. Their

[1] Corresponding author

experiment focused on programming efficiency, error frequency, and types of errors. In contrast, our study targeted younger first-year Computer Science students at our university and was conducted on a larger sample size. In another study [2], authors compare three different kinds of editor including textual with code-completion, projectional one and form based.

2 Languages and tasks

Working with younger and less experienced programmers requires careful selection of programming environments and languages for the experiment. Additionally, it required choosing tasks that students could effectively complete during the experiment. These tasks had to strike a balance – they couldn't be too complex, as that would affect programming times, nor could they be too lengthy, as students wouldn't be able to complete them within a single class session. Furthermore, tasks couldn't be too brief, as they wouldn't yield to meaningful conclusions.

2.1 Languages

In our endeavor to work with MPS, we undertook an exploration of its baselanguages [4] to determine which would be suitable for our experiment. Together with a programmer boasting approximately eight years of experience in the field, we experimented with several languages, including C# [8], Java baselanguage, HTML, CSS, Javascript, and others. Among these, only the Java baselanguage editor exhibited the required level of suitability for our experiment. Given the students' lack of experience with Java, we continued our search and identified the Kaja language as a viable alternative. Kaja bears resemblance to the Karel Robot language, with which students typically engage in their first year. While Kaja possessed a commendable editor, some modifications were necessary. We accordingly tailored the editor to closely align with the Karel library in C, developed by Miroslav Biňas at TUKE. The disparities between the two included the type declaration for the main function (*void* instead of *int*), the invocation of the main function (which is unnecessary in C, unlike MPS), the *turn_on*() function (which lacked a map parameter in MPS), the absence of the $turn\_off()$ function in MPS, and the *return* 0; statement (not present in MPS). Snippets of both implementation could be found in Listings 1 and 2.

Listing 1 Snippet - Karel in MPS after Editor correction.

```
include < task1kw >
main();


void main(){
 turn_on();
 while ( front_is_clear() ) {
  step();
  turn_left();
  check_beepers();

  while ( facing_south() && front_is_clear() ) {
   step();
   check_beepers();
  }
```

```
   turn_around();
   go_back();
  }
 }
```

Listing 2 Snippet - Karel in C.

```
#include <superkarel.h>

int main(){
  turn_on("task1.kw");

  while(true){
    turn_left();

    check_beepers();
    while (facing_north() && front_is_clear())
    {
      step();
      check_beepers();
    }
    turn_around();
    go_back();
  }

  turn_off();
  return 0;
}
```

As the text editor for our experiment, we selected VS Code, as it is widely utilized by the majority of programming students at our university. Furthermore, acquiring all necessary plugins for the experiment was straightforward due to its popularity and extensive plugin ecosystem.

2.2 Tasks

We had a total of four tasks for the experiment, each focusing on different aspects. Task 1 involved code transcription, requiring students to transcribe a 48-line code containing while loops and if conditions. Task 2 required students to correct all syntactic errors in source code, 10 in total. Task 3 tasked students with programming an algorithm, either independently or with hardcoding, to ensure Karel traversed the entire map. Finally, Task 4 focused on code refactoring, wherein students had to shorten the code in the main function by writing three new functions: *turn_left*, *horizontal_line*, and *vertical_line*.

Prior to the main experiment, four pilot mini-experiments were conducted to identify shortcomings in the tasks and assess whether a student could complete both text and projectional editor experiments within a single class session. Based on the data obtained from these mini-experiments, tasks were revised accordingly. It was also determined that it was not feasible for students to complete experiments in both types of editors within a single class session. Additionally, it was concluded that students intending to program in the projectional editor MPS needed some prior exposure to MPS before the main experiment. Among several proposals, it was decided to prepare Task 1a, where students had to transcribe the code,

rebuild the project, and then execute the task. This proposal was ultimately implemented in the experiment. Alongside the tasks, students received instructions briefly describing MPS and its features, such as code completion, in addition to the information provided during our explanation of Task 1a.

3 Experiment

The experiment took place during Programming classes. We successfully conducted the experiment across five study groups, with voluntary participation in two of them. On Monday at 7:30 AM, 11 students participated, while only 5 students attended at 9:10 AM; both sessions involved programming in the text editor VS Code. At 10:50 AM, attendance was mandatory for the experiment conducted in MPS, with 22 students participating.

On Tuesday, the experiment continued at 7:30 AM in MPS, with 21 students participating, followed by another session at 9:10 AM in the text editor, which attracted a total of 24 students.

Study groups were formed in the beginning of the study year based on the alphabet order of students' surnames. Therefore, we believe that students are mostly uniformly distributed based on their programming experiences in our five study groups.

3.1 Issues

During the experiment, several issues arose. The first concerned OBS recording in Linux, which malfunctioned multiple times during the initial sessions with the first two groups. Recordings either froze on a single frame or stopped altogether after completing certain tasks. We managed to resolve this issue by the second day of the experiment.

The second issue pertained to Task 2, where students were required to correct syntactic errors. In the projectional editor MPS, certain syntactic errors resulted in the deletion of entire functions, effectively transforming the task from error correction to completion. Unfortunately, we were unable to resolve this issue, so we reloaded the task on three computers and manually reintroduced the syntactic errors on each one individually.

Another issue we encountered necessitated adjustments to the time measurement for tasks. This issue stemmed from differences in computational resource consumption between the text and projectional editors. In the text editor, it's possible to adjust the speed of Karel's movement on the map. We attempted to adjust Karel's speed in MPS as well, which worked up to a certain point. Beyond that point, reducing the pause between steps did not accelerate Karel on our computers. To address this, we attempted to run Karel on a more powerful computer, where reducing the pause did indeed speed up the rendering. For the experiment, we maintained the pause value at 50 milliseconds, and a noticeable difference in Karel's movement on the map was observed across all 21 computers.

4 Results

Before reviewing the recordings obtained from students, we established specific rules for calculating time, compilation frequency, and error counts. Time calculation was divided into two groups. In the first group, consisting of Task 1, Task 3, and Task 4, we measured time from the first keystroke in the editor to the successful compilation of the program – no waiting was required as the speed of Karel's movement varied. For Task 2, we measured time from file opening to successful compilation.

We simplified the counting process by focusing on syntactic errors rather than logical errors. This resulted in disparities in the number of errors and compilations. Students programming in the projectional editor MPS avoided syntactic errors due to its syntax checking feature. Consequently, our focus was on the number of syntactic errors students made in the text editor.

The compilation count varied because students made syntactic or logical errors, and some of them programmed incrementally. For instance, they would write a few lines of code and then verify whether the program functioned correctly. In Task 3, they would determine additional commands to program.

4.1 Task1

During the experiment, issues arose that resulted in losses of recordings. In Task 1, we obtained a total of 72 out of 83 recordings. Students programming in MPS achieved an average time of 5 minutes and 52 seconds for the first task, which is 4 minutes and 28 seconds, or 40 percent, faster than students using the text editor. The disparity between the best times achieved in the projectional and text editors also revealed a 44 percent advantage for the projectional editor, with the worst time in MPS being even 47 percent faster than in the text editor.

The average number of compilations was 1.24 for the projectional editor and 1.97 for the text editor, while the average number of errors in the text editor was 1.17.

Table 1 Results Task 1.

Task 1	MPS	KinC	MPS1	MPS2	C1	C2
Subjects	41	31	21	20	7	24
Ø time	5:52	10:20	5:42	6:02	13:38	9:22
MIN time	2:04	3:44	2:04	3:32	8:39	3:44
MAX time	11:54	22:47	11:54	11:35	22:47	19:58
Ø errors	0	1.17	0	0	1.60	1.08
Ø compiles	1.24	1.97	1.33	1.15	2.00	1.96

1) MPS – complete MPS group.
2) KinC – complete Karol in C group.
3) MPS1 – Monday MPS group.
4) MPS2 – Tuesday MPS group.
5) C1 – Monday group Karol in C.
6) C2 – Tuesday group Karol in C.
7) Subjects - number of students participated in a group.
8) Ø time – the average time it took students to complete the task.
9) MIN time – the shortest time solving the task.
10) MAX time – the longest time solving the task.
11) Ø errors – the average number of errors in a given task.
12) Ø compiles – average number of compiles of a given task.

4.2 Task2

In Task 2, we obtained only 29 out of 83 recordings. This task was not entirely balanced, as MPS highlights all syntactic errors, and with the cursor appropriately positioned, activating auto-completion would either automatically correct the error or suggest a few options for completing the code at that point.

Despite this imbalance, we evaluated this task, and the average time in the projectional editor was 1 minute and 11 seconds compared to 3 minutes and 1 second in the text editor, resulting in a 76 percent advantage for the projectional editor. However, it is worth noting that from later observations, we discovered that three students who completed this task in MPS had slightly above-average completion times for all tasks in the projectional editor.

The average number of compilations was one in MPS because errors were highlighted, eliminating the need to compile the program to identify errors, as was the case with the text editor. We did not calculate the average number of errors in this task because students were correcting errors rather than creating them.

Table 2 Results Task 2.

Task 2	MPS	KinC	MPS1	MPS2	C1	C2
Subjects	3	26	0	3	3	23
Ø time	1:11	3:01	0	1:11	2:59	3:01
MIN time	0:54	1:35	0	0:54	1:50	1:35
MAX time	1:38	9:24	0	1:38	4:55	9:24
Ø compiles	1.00	2.08	0	1.00	1.67	2.13

1) Rows and columns use the same notation as it is in the Table 1.

4.3 Task3

In Task 3, students encountered the most challenges, despite the task requiring only for Karel to traverse the entire map, with the option of utilizing hardcoding. This is evident primarily from the average number of compilations, which was significantly higher compared to other tasks, standing at 1.83 in the projectional editor and even 3.22 in the text editor. The average completion time for the task in MPS was 5 minutes and 25 seconds, whereas in the text editor, it was 7 minutes and 36 seconds, representing a difference of only 29 percent. The disparity between the best time in the projectional editor and the best time in the text editor was 43 percent. The worst time in the text editor was 31% slower than in the projectional editor.

Table 3 Results Task 3.

Task 3	MPS	KinC	MPS1	MPS2	C1	C2
Subjects	42	27	21	21	3	24
Ø time	5:25	7:36	5:17	5:34	5:54	7:48
MIN time	1:24	2:28	1:33	1:24	2:34	2:28
MAX time	13:41	20:02	13:39	13:41	8:30	20:02
Ø errors	0	1.15	0	0	0.33	1.25
Ø compiles	1.83	3.22	1.90	1.76	1.33	3.46

1) Rows and columns use the same notation as it is in the Table 1.

4.4 Task4

In Task 4, students were required to refactor code. We obtained 69 out of 83 recordings from students for this task. The average completion time for this task was 5 minutes and 3 seconds in the projectional editor and 5 minutes and 59 seconds in the text editor, resulting

in a mere 15 percent difference. Once again, in this task, the difference between the fastest student in the text and projectional editors was just under 50 percent, while for the longest completion times, this difference exceeded 52 percent. Students working in the text editor made an average of 0.85 errors, which is the lowest number among all three tasks where errors were counted.

Table 4 Results Task 4.

Task 4	MPS	KinC	MPS1	MPS2	C1	C2
Subjects	41	26	21	20	2	24
Ø time	5:03	5:59	5:02	5:05	3:33	6:11
MIN time	1:30	2:57	1:30	2:39	3:23	2:57
MAX time	11:07	23:31	9:16	11:07	3:42	23:31
Ø errors	0	0.85	0	0	0	0.92
Ø compiles	1.15	1.81	1.14	1.15	1.00	1.88

1) Rows and columns use the same notation as it is in the Table 1.

4.5 Summary

Due to the issues encountered during the experiment, the number of obtained data points reduced from 83 to 28 for completing all 4 tasks. Table 5 presents the averaged results from all 4 tasks or, in the case of columns 3 and 4, from Task 1, Task 3, and Task 4. Columns 1 and 2 in this table are not entirely reliable since only 3 students completed all 4 tasks in the projectional editor. Therefore, we averaged the results from Task 1, Task 3, and Task 4, which are displayed in columns 3 and 4. From these results, we observe that students completed tasks in MPS on average in 16 minutes and 16 seconds, which is nearly 30 percent faster than students programming in the text editor. The fastest completion time in MPS was again over 60 percent faster than in the text editor, while the difference between the slowest times favoured the projectional editor MPS by 36 percent. The average number of errors for the entire experiment was 3.15 for the group of students programming in the text editor, and the average number of compilations for Task 1, Task 3, and Task 4 in the projectional editor was 3.42, which is over half less than in the text editor.

Table 5 Overall Results.

	MPS	KinC	MPS T1,T3,T4	KinC T1,T3,T4
Subjects	3	25	40	26
Ø time	13:16	25:16	16:16	23:06
MIN time	9:40	15:14	5:07	12:48
MAX time	16:24	44:01	27:06	42:26
Ø errors	0	3.15	0	3.15
Ø compiles	4.60	9.16	3.42	7.08

1) Rows and columns use the same notation as it is in the Table 1.

4.6 Questionnaire

The experiment involved two questionnaires, each containing different questions along with four common questions. These common questions pertained to the tasks of the experiment and focused on the text editors students use at home for programming. From the questions

related to the tasks, it emerged that all tasks were perceived as easy for various subjective reasons. A large group of students had no difficulty with any task, while a smaller group struggled with Task 4, which they felt was insufficiently explained. The majority of students use VS Code for programming at home, while the remaining students use either Clion or Vim.

Among the drawbacks of MPS mentioned by students were mainly its complex interface or issues with code deletion, where incorrect cursor settings could delete a larger portion of code compared to a text editor. They also mentioned issues such as the speed of rendering Karel on the map, code copying, inability to select code with a mouse, and so on. One of the drawbacks concerned the complexity of typing "&&" in the Karel language editor.

On the other hand, the advantages of MPS mentioned included fast code typing, automatic code formatting, code completion, interesting text selection features (Ctrl + arrows), quick copying, or the inability to make mistakes. From these responses, it was clear that while some students found the projectional editor suitable and comfortable to work with, others did not feel the same even after completing all four tasks. A significant portion of students noted that working in MPS was initially challenging, but they eventually got used to it, and programming became easier for them.

One question asked whether students would prefer to program in a projectional editor or a text editor. From the questionnaire, it emerged that 3 students would prefer to program in a projectional editor, 3 students were undecided, and the rest would prefer to continue programming in a text editor.

5 Conclusion

We presented an experiment comparing text and projectional editors in terms of code writing speed and error rate. The experiment consisted of several stages. Initially, we focused on MPS as a projectional editor and its baselanguages. We explored various options and selected the most suitable language for the experiment, which turned out to be the Kaja language in MPS. Then, we examined different variants of the Karel Robot language in the text editor and selected a library that was created and used at our institution.

After selecting the language and library for the text editor, it was necessary to adjust the editor for the Kaja language in MPS. While we mostly succeeded in this endeavor, there were some minor differences that did not significantly affect the experiment's results. Subsequently, we devised tasks for students to program, from which we could obtain data for comparing the editors. These tasks, totaling four, were designed for the measurements we aimed to conduct.

To improve and refine the experiment's shortcomings, we conducted four mini-experiments, addressing questions and identifying task deficiencies. During these mini-experiments, we adjusted the tasks for the final version provided to students for the main experiment. The final version of the tasks included a tutorial and Task1a for the group using the projectional editor, which served as an introduction to MPS. However, due to issues encountered, we did not obtain all the information possible from the experiment.

Upon processing the data, we found that students programming in MPS during the experiment were 30 percent faster than those using the text editor. The group using MPS made no syntactic errors, reflected in the number of compilations. Conversely, students using the text editor made such errors, leading to a higher number of compilations compared to the MPS group. Despite this, an average of 3.42 errors across three tasks (a total of 85 lines of code) for first-year computer science students at our institution is considered acceptable. We believe this experiment sheds light on projectional editors and their clear advantage in programming speed, as evident from our findings.

Our experiment was not targeted on the level of understanding nor the effectivity of teaching a student programming or a programming language. From our experiment we could not simply conclude that a student will benefit from teaching a new programming language with projectional editor. However, projectional editor could help a student focus on concept not the notation, what could lead to some advantages. Following studies may provide deeper insides.

We consider it challenging to further extend this experiment. While there are numerous experiments that could be conducted, it would require a larger pool of MPS users. However, this presents a challenge as MPS has a small user base, affecting the variety of languages and language editors available in the tool. Similarly, it also affects the availability of resources dedicated to this topic.

References

1 Thorsten Berger, Markus Völter, Hans Peter Jensen, Taweesap Dangprasert, and Janet Siegmund. Efficiency of projectional editing: A controlled experiment. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 763–774, 2016. doi:10.1145/2950290.2950315.

2 Sergej Chodarev, Matúš Sulír, Jaroslav Porubän, and Martina Kopčáková. Experimental comparison of editor types for domain-specific languages. *Applied Sciences*, 12(19):9893, 2022.

3 Veronique Donzeau-Gouge, Gerard Huet, Bernard Lang, and Gilles Kahn. *Programming environments based on structured editors: The MENTOR experience*. PhD thesis, Inria, 1980.

4 Vadim Gurov. Baselanguages in mps. URL: https://youtrack.jetbrains.com/articles/MPS-A-13697566.

5 Louis-Edouard Lafontant and Eugene Syriani. Gentleman: a light-weight web-based projectional editor generator. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 1–5, 2020. doi:10.1145/3417990.3421998.

6 Markus Voelter and Sascha Lisson. Supporting diverse notations in mps'projectional editor. In *GEMOC@ MoDELS*, pages 7–16, 2014. URL: https://ceur-ws.org/Vol-1236/paper-03.pdf.

7 Markus Voelter, Janet Siegmund, Thorsten Berger, and Bernd Kolb. Towards user-friendly projectional editors. In *International Conference on Software Language Engineering*, pages 41–61. Springer, 2014.

8 Tomáš Eliáš; Roman Firment; Jakub Saksa; Martin Wirth; Dalibor Zeman. C# base language for mps. URL: https://www.ksi.mff.cuni.cz/sw-projekty/zadani/cs4mps_spec.pdf.