Evaluation of the impact of using an IDE to learn an object-oriented programming language

Thomas Gay - 74672

September 28,2018

1 Introduction

According to the *TIOBE index for September 2018*, five of the top ten programming languages are object-oriented. The job website Indeed.com provides *The 7 Most In-Demand Programming Languages of 2018* in which four are object-oriented and the two first ones are Java and Python. This sector is increasing and offers many opportunities. Faced with this strong job offer, many people want to learn object-oriented programming languages. Before writing the first lines of code, people have to set up their working environment and chose a tool to help them to learn. This paper is trying to quantify the impact of such a tool, in particular integrated development environments (IDEs).

2 Scope

In this experiment, two type of tools are compared. When people want to teach or learn an object-oriented programming language they have to choose an environment, i.e. a simple text editor or an integrated development environment (IDE). The first one is very simple to install and use but it doesn't provide any help. IDE is a bit more complicated to use because of the number of features, it takes times to control them but they provide more help and it is easier to find errors during the programming process.

Objective. The purpose of the experiment is to compare the two tools and find which one is more efficient to learn an object-oriented programming language like Java or C++. Learning a programming language means to have basic skills to be able to implement short application in a reasonable time.

Object of study. The object of study is persons without experience in programming. Their ability to learn software sciences and programming and their determination are objects of this experiment.

Quality focus. The main parameters studied in the experiment are the time to create and implement an application, the number of errors, and the code optimization through the number of lines of code (LOC). About the last criteria, lines of code are relevant to know in how many steps a subject achieves its goals: less code it is less time spent to programming and more to thinking about the best solution.

Context. The experiment is focused on the comparison between text editor and IDE to learn an object-oriented programming language. The experiment includes 40 persons who haven't skills in programming. They are volunteers to participate. That means all participants have palatability to programming and with this lack of randomization of subjects the experiment is not fully into control. Participants have followed the same course of object-oriented programming. Four environments (two text editors and two IDEs), all under Linux Ubuntu, were used during the experiment: Gedit, GNU Emacs, IntelliJ IDEA Community and Eclipse Oxygen for Java. In both categories we use two different tools to prevent variation from tool efficiency or inefficiency. Then they were divided into groups to learn the basis of each tool.
Of course, it is an off-line experiment, in that way the process of learning and evaluating subjects is under control.

3 Experimental design

The experiment attempted to answer the following questions:

1. Do IDE reduce the time of implementing programming tasks? This question is, probably, the most important one to compare IDE and text editor. The goal is to learn faster an object-oriented programming language and create software and applications. With which one of these tools is it faster?
2. Do IDE increase the correctness of the delivered solution? The correctness is about the number of errors in a solution and the degree of the errors.
3. Do IDE increase the quality of the solution submitted? This quality can be measure with the number of lines of code which provide hint about the code optimization.

3.1 Variables

Independent variables. This experiment has two main independent variables. The use of an IDE or a text editor. The use of documentation for IDE features and Java can be considered as external factors.

Dependent variable. The experiment has four dependent variables:

- T: Time to implement a short application excluding environments setting.
- LOC : Number of lines of code.

The correctness of a solution is measured by:

- C: Number of submissions of a solution with a fault.

3.2 Hypotheses

With this experiment, the null hypothesis says that learn object-oriented programming language is not better with an integrated development environment than a text editor. On the contrary, the alternative hypothesis express the fact that to learn with an IDE is better in term of implementing time, number of errors and code optimization.

Table 1: Tested Hypotheses

Dependent variables	Null hypothesis (H_0)	Alternative hypothesis (H_1)
Time to implement application	T(IDE)$\geq$T(TE)	T(IDE)$<$T(TE)
Num of submissions of solution with a fault	C(IDE)$\geq$C(TE)	C(IDE)$<$C(TE)
Num of lines of code	LOC(IDE)$\geq$LOC(TE)	LOC(IDE)$<$LOC(TE)

3.3 Subject

Subjects were recruited among several university in Lyon, France via a request oriented for student from any kind of studies. The requirements were to want to learn programming, have no skills in programming language and have the baccalaureate (it is the degree which people obtain at the end of high school). The request specified a range of time and ask for several questions about the candidate. Subjects come from Business and Economics university, Sciences and Technology university and Engineering school. Thanks to the answers of the subject's background, it was possible to divide them into groups were the characterizes are equivalents.

3.4 Experiment Design

To summarize, the use of an IDE or a TE are the independent variables. The dependent variables have been chosen (Time to implement, Number of lines of code, number of errors).

Randomization. The selection of the subjects will be representative of students in bachelor or master degree with no knowledge of programming. The assignment to each treatment (using an IDE or a TE) is selected randomly. It is a completely randomized design experiment.

Blocking. To avoid to compare two tools but to type of tool. To minimize the effect of the efficiency (or inefficiency) of an environment, the subject will be divided in 4 groups: 2 will do the experiment with a different IDE (Eclipse, IntelliJ) and 2 with 2 different text editors (Gedit, Emacs).

Balancing. The experiment use a balanced design, each group will be composed by 10 persons. In that way, there are 20 persons in each category of tool.

3.5 Instrumentation

Experiment object. The objects of the experiment is the code produce by the subjects and the submission of this code.

Guidelines. All the subjects will have the same theorical formation in Java. After, they will have a description of the features of their tools respectively.

Measurement instruments. In this experiment it is important to validate the subject's submission. That's way there is program which runs tests to be sure all the functional requirements are conformed to the specification. Another tool, called CLOC, will count the number of lines in the program files.

3.6 Data collection

The experiment will begin with a three days of intensive Java courses. All the subjects will be together and learn in the same way. After, they will be divided into the groups and each group will have a two days training of the features of their tool. In the end, the subjects have to implement a naval battle game with a 5x5 board, one 1x3 boat and two 1x2 boat. This project will be divided in some task describe below. For each task, the subject should submit his source code. This one will be analysed to count the number of lines and the number of errors. At the end of the experiment, subjects will answer to a questionnaire about their feeling during the experiment.

3.6.1 Task 1

The first one is a simple task. The subjects have to create naval battle game in command line. This version includes only one player who play against the computer. The computer shoots randomly to the enemy board. An example of the result expected (B = Boat, M = Missed, H = Hit)

```
$> shoot(1,2)
Missed!
Enemy missed!
Your board:

| B | B | B |   |   |
|   |   |   |   | M |
| B | B |   | B |   |
|   |   |   | B |   |
|   |   |   |   |   |
```

```
Enemy board:
|   |   |   |   |   |
| M |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |

$> shoot(5,5)
Hit!
Enemy hit!
Your board:
| B | B | B |   |   |
|   |   |   |   | M |
| B | H |   | B |   |
|   |   |   | B |   |
|   |   |   |   |   |

Enemy board:

|   |   |   |   |   |
| M |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   | H |
```

The program has to know who has won the game.

3.6.2 Task 2

This task is more about an algorithmic problem. Subjects have to implement a more intelligent computer. To validate this task, the artificial intelligent must have a win rate of 95% against a random intelligence.

3.6.3 Tasks 3

The last task is to create a user interface with Java FX. The interface must display only the boards of the player. At the beginning, the player arrives to a menu where he can select a difficulty ("easy" = random AI or "medium" = AI implemented in task 2).

3.7 Analysis procedure

The analysis procedure includes a quantitative analysis and a short qualitative analysis. The hypotheses will be tested with the quantitative data. The qualitative data increase the comprehension of the results.

3.7.1 Quantitative analysis

The analyses of the dependent variables will be performed for each task and across all tasks. It will be performed descriptive statistic and t-test for all the dependent variables (T, LOC, C and C').
The risk level to reject or no the null hypothesis is 0.05. For each set of tests, we will provide the calculated p-value. Thanks to the number of subjects it is possible to do an Analysis of Co-variance, also called ANCOVA for the time, the LOC and the correctness for all the tasks and subjects. We have a total 120 data points for each dependent variable (40 subjects and three tasks).

3.7.2 Qualitative analysis

The analysis of the answer the questionnaire about the feeling may help us to understand the results of the experiment and the increase the strength of the results. The questionnaire takes around 50 minutes to answer all the questions. For all the questions there is a nominal choice (like Does IDE help you to create user interface? Answers can be: Not at all, Few times, Many times, All the time). In that way, for each question, there are a distribution of answers which can confirm the results of the statistical experiment.

3.8 Evaluation of validity

As it says before, after each submission the code will be analyse and test. If a submission is not sufficient or doesn't fit with the specifications, the subject has to do the necessary changes (the time to do it will be added to the initial time). For example, to test the *Task 2* the subject's AI will play around 1 000 games against a random AI, and the subject's one has to win more than 950 times. For *Tasks 3*, the subjects have an GUI specification with example of what it is expected. The subjects will be timed for each work. These requirements are important to be able to compare the final solution in term of effort. The time allocated for each task is not fixed because some of them can be faster or slower than the others.

Each subtask will be rated as *acceptable* or *unacceptable* according to the following criteria. Will be unacceptable :

- duplication of existing code
- illogical java class
- non object-oriented programming

4 Analysis

4.1 Descriptive statistics

We will present the results from the experiment described in the *Experimental Design*. Recall that we are testing three hypotheses:

- The time (T) to develop the application
- The correctness (C) of the solutions, i.e. number of errors per submissions
- The code optimization in term of number of lines of codes (LOC)

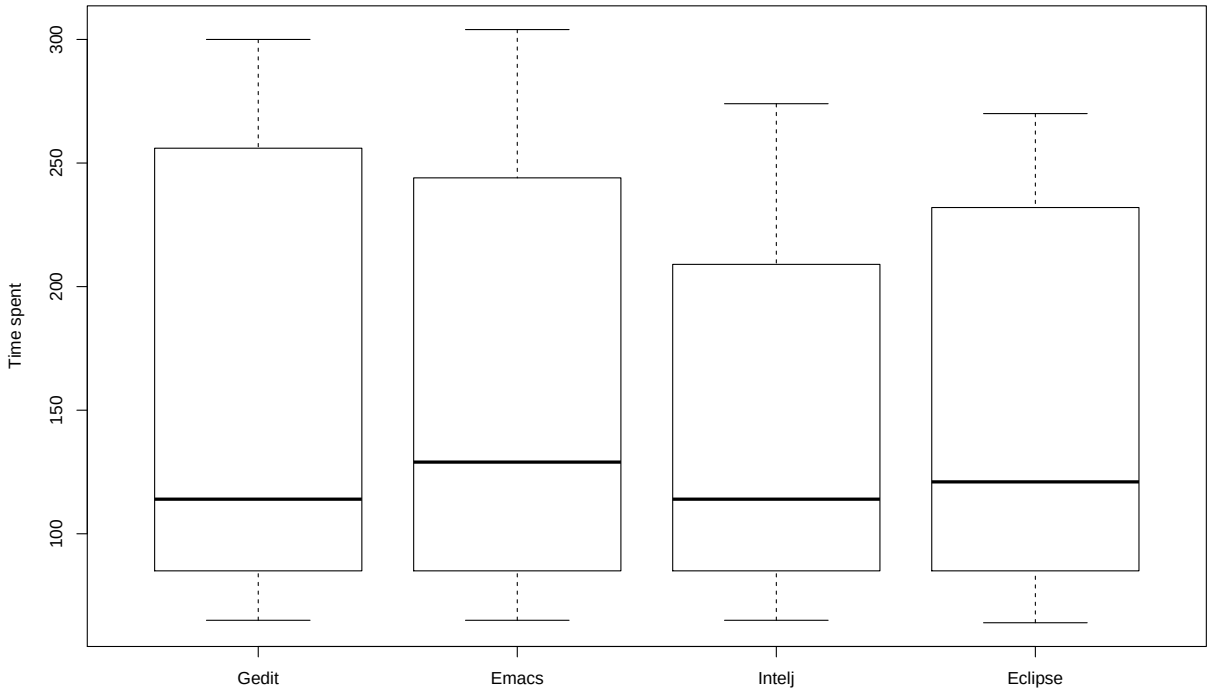
4.1.1 Time

Table 2 shows the descriptive analysis for the time which is expressed in minutes. We did an analysis for each task and across the tasks, respectively. For the time, we calculated the mean, the median, the standard deviation, the minimum, and maximum value for both two treatments.
In term of time, the TE group spent 12.2 percent on average more time in total. The mean is lower for the first and third tasks for the IDE group. For the second task, the TE group spent, on average, less time than the IDE group. The maxima and the minima for the IDE group are always smaller than the ones of the TE group.
Figure 1 shows the time across all the tasks for each tool.

Table 2: Descriptive Statistics : Time

	Mean		Median		Standard deviation		Min		Max	
	TE	IDE	TE	IDE	TE	IDE	TE	IDE	TE	IDE
Task 1	262.3	233.6500	265	234.0	24.71543	25.41917	201	190	304	274
Task 2	82.1	82.8500	76	84.0	14.35967	11.15572	65	64	107	103
Task 3	123.0	111.7000	115	116.5	31.77718	21.29517	78	74	181	157
All	155.8	142.7333	115	116.5	81.48054	68.83482	65	64	304	274

Figure 1: Time spent for all the tasks



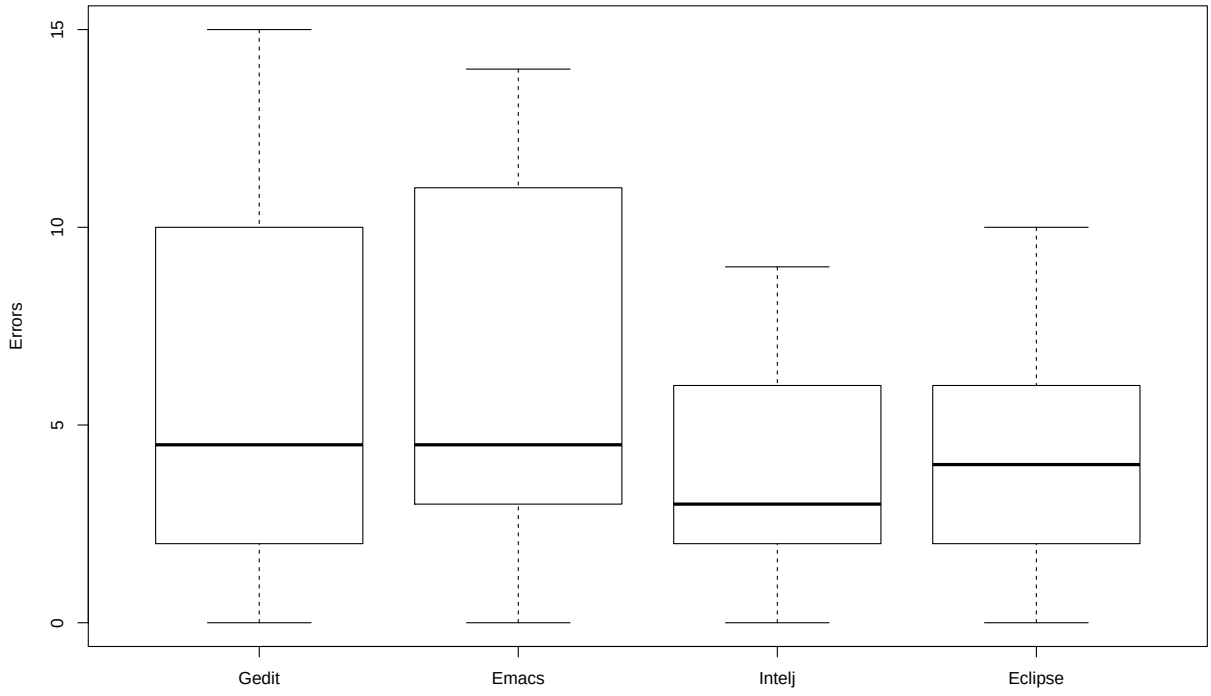
4.1.2 Correctness

Table 3 shows the results for the correctness. On average, TE subjects make 2.19 errors more than the subjects of the IDE groups. The minimum and maximum value for the second task are quite similar between the two groups. However, the gap in the last task is bigger, with a maximum of 15 errors for the TE group against 10 errors for the IDE subjects.
Figure 2 is a boxplot of the errors made by the subjects of each tool.

Table 3: Descriptive Statistics : Errors

	Mean		Median		Standard deviation		Min		Max	
	TE	IDE	TE	IDE	TE	IDE	TE	IDE	TE	IDE
Task 1	4.60	3.400000	5.0	3.5	3.3150375	1.788854	0	0	10	7
Task 2	2.45	2.150000	2.5	2.0	0.8870412	1.225819	1	0	4	4
Task 3	11.55	6.500000	11.5	7.5	2.5438264	2.460210	3	2	15	10
All	6.20	4.016667	4.5	3.0	4.6058180	2.619947	0	0	15	10

Figure 2: Errors for all the tasks



4.1.3 LOC

Table 4 shows the results for the correctness. On average, TE subjects write 6.66 lines less than the subjects of the IDE groups. The minimum and maximum value can be very different for task 1 and 2.
Figure 3 to 5 shows the number of lines for each tool and for each task. The number of lines total is not relevant here, we will discuss it in *Interpretation*.

Table 4: Descriptive Statistics : LOC

	Mean		Median		Standard deviation		Min		Max	
	TE	IDE	TE	IDE	TE	IDE	TE	IDE	TE	IDE
Task 1	749.2000	710.55	757.5	707.0	58.385831	45.738933	641	607	841	790
Task 2	43.2500	47.15	43.5	47.0	5.210061	1.843195	35	43	55	51
Task 3	192.7000	247.45	195.5	247.5	21.849003	19.794936	148	207	237	283
All	328.3833	335.05	195.5	247.5	308.366615	281.595540	35	43	841	790

Figure 3: LOC for task 1

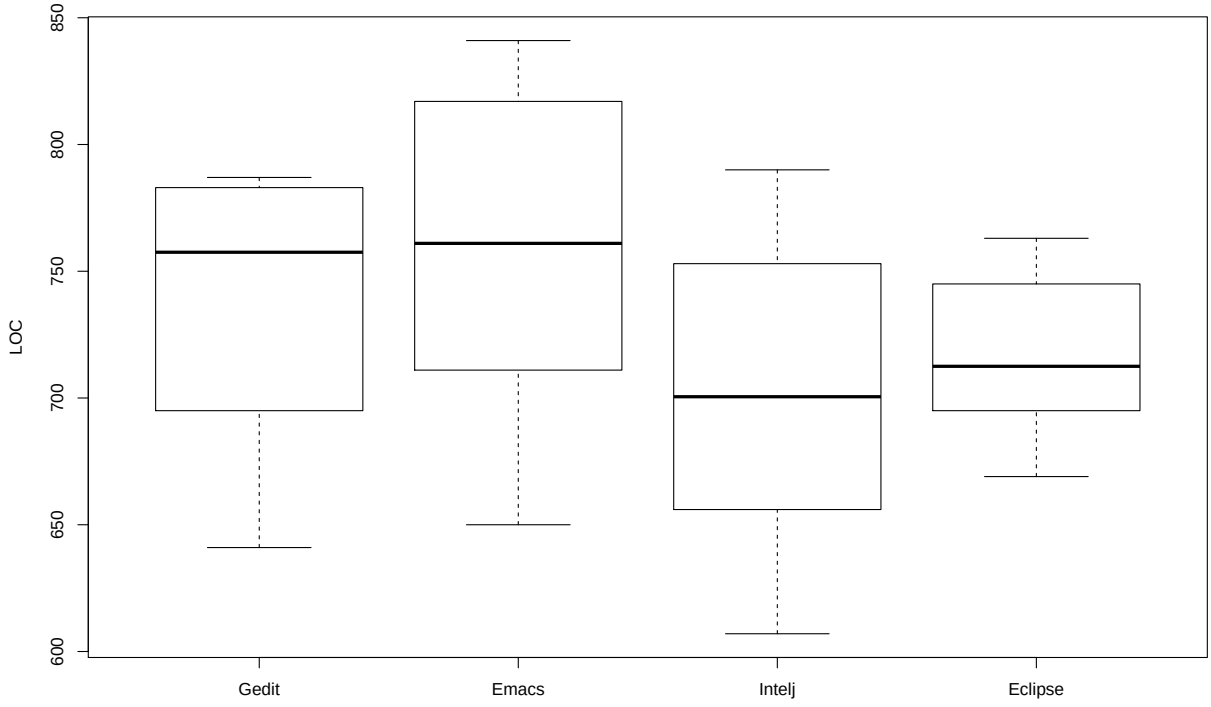


Figure 4: LOC for task 2

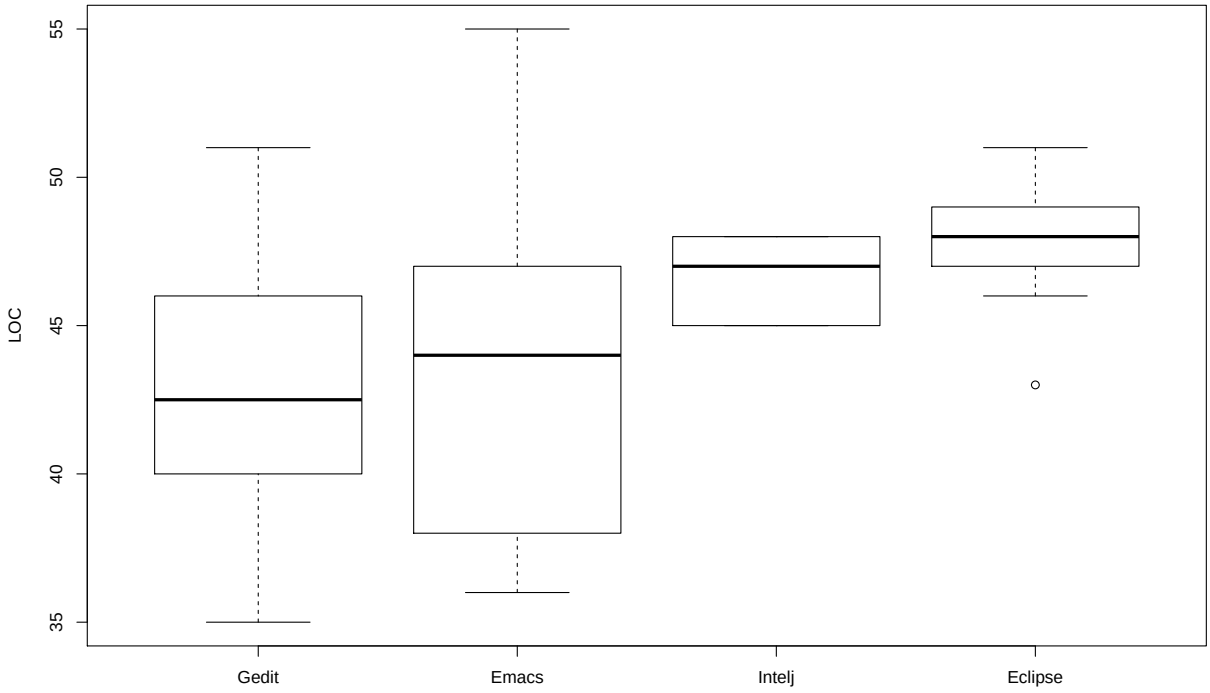
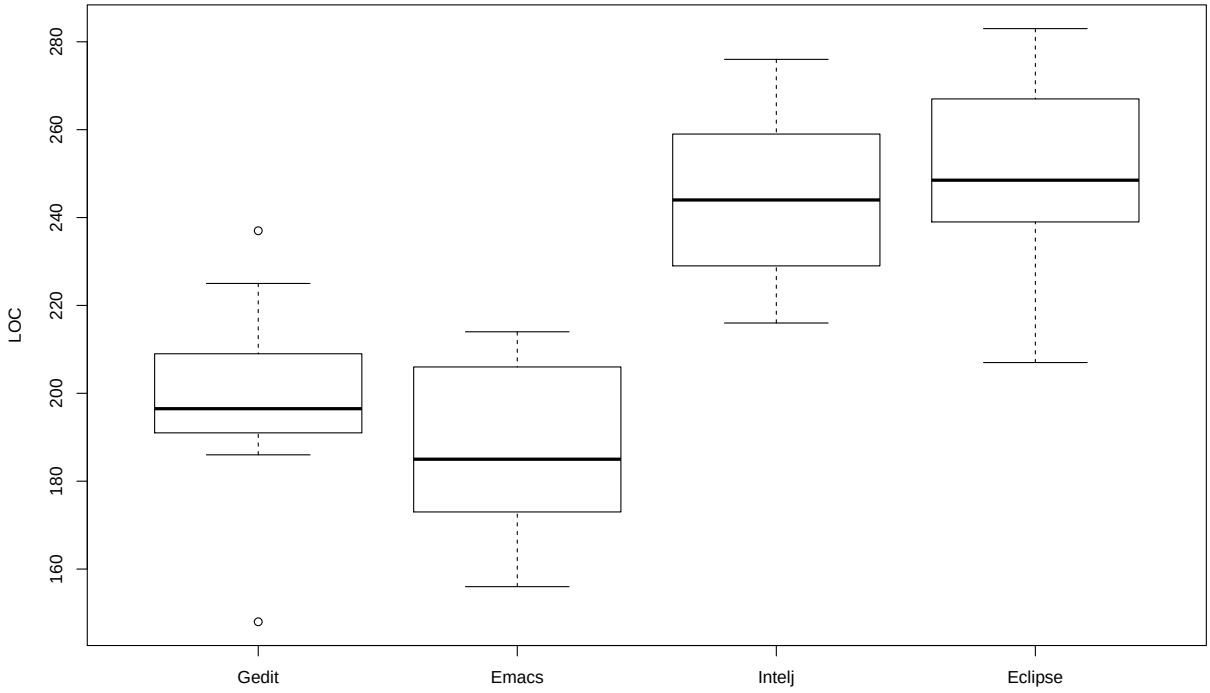


Figure 5: LOC for task 3



4.2 Hypothesis testing

4.2.1 Time to implement application - H_0:T(IDE)≥T(TE)

For the time, we will test the hypothesis for all the tasks and for each task because of the difference between the TE group and the IDE one is changing from one task to another.We will use the t-test.

```
##
## All the task       [,1]
##      p-value  0.1723323
##      t-value -0.9489004
```

```
##
## Task 1             [,1]
##   p-value  0.0004359605
##   t-value -3.6138844165
```

```
##
## Task 2         [,1]
##   p-value 0.5726517
##   t-value 0.1844557
```

```
##
## Task 3           [,1]
##   p-value  0.09775308
##   t-value -1.32108469
```

4.2.2 Number of submissions of solution with a fault - H_0: C(IDE)≥C(TE)

To test the hypothesis of the correctness, we use a *Wilcox* test known as *Mann-Whitney* test. It is not necessary to test for each task because og the homogeneity of the results and the differences between the two groups.

```
##
## Correctness        [,1]
##     p-value 1.28199e-02
##     w-value 1.37700e+03
```

4.2.3 Number of lines of code - H_0: LOC(IDE)≥LOC(TE)

For the number of lines of code, we are using the same approach than for the time. You can see below the results of these tests.

```
##
## All the task      [,1]
##      p-value 0.5491016
##      t-value 0.1236598
```

```
##
## Task 1           [,1]
##   p-value  0.01275529
##   t-value -2.33048032
```

```
##
## Task 2         [,1]
```

```
##   p-value 0.9978414
##   t-value 3.1559500
##
## Task 3        [,1]
##   p-value 1.000000
##   t-value 8.304905
```

5 Interpretation

5.1 Evaluation of results

In this section, we will discuss the results of the descriptive statistic and the test of the hypotheses. We will talk about the results across the tasks and for each task.

First, if we have a look at figure 1, we can see that the time is quite similar between the tools. However, in table 2, we can observe that for the first and third task, the IDE group took less time to perform these tasks. For the second task, the time needed is the same for both groups. The hypothesis testing shows that we can assume that for task 1 T(IDE) is smaller than T(TE) with a risk of 5% (p=0.0004). In the same way, for task 3 we can say that T(IDE) is smaller than T(TE) with a risk of 10% (p=0.097). We can understand these results because IDEs help people to browse through classes and file and they lost less time to verify the functions' names and classes' name. These features are pointed out by the questionnaire filled by the subjects at the end of the experiment. They say that the suggestions of the IDEs are very helpful. Now, we can focus on task 2. it is impossible to say if T(IDE) is smaller, greater or equal to T(TE) (p=0.57). The main cause of this result is that task 2 is an algorithmic problem. Such a problem can be solved only by reflexion process and the tools are useless in this context. With the descriptive analysis, we guess that the time needed is the same for the IDE group and the TE group.
Now, we will discuss the results about the correctness. The descriptive statistic and the hypothesis testing give the same result: with a risk of 5% (p=0.013) we can say that IDEs help people to produce software with fewer errors than the ones who code with a text editor. The questionnaire gives us a possible answer. IDEs detect many errors before compiling or testing code and help users with hints to resolve these errors. This help is very useful for task 3 because this task is about a user interface and in this kind of task, there is a lot of hidden errors, that means an error which doesn't block the program but which is not correct.
Finally, let's have a look at the number of lines of code (LOC). The global number of lines is not relevant here because the tasks are too different from the LOC approach. For task 1, we can say that the number of lines of code needed is smaller for the IDE group than the TE. Here, again, the suggestions of IDEs are helpful, for example, they propose to use a function which can do the job instead of doing the function by yourself. About task 2, we can say nothing about our hypothesis due to the nature of this task. Like for task 1, an IDE can't help for an algorithmic problem in term of time or number of lines. The last result is a bit surprising. In fact, the LOC needed for this task seems to be smaller for the text editor group. thanks to the questionnaire and the analysis of the code, we conclude that IDEs add lines of code automatically. IDEs try to respect *good practices*, that's why they add lines.

To conclude, we can say that IDE helps people to learn the basics of an object-oriented language like java in term of errors, time and quality. However, to learn how to resolve algorithmic problems, the tool doesn't have any impact on the process.

5.2 Limitations of study