Benchmarking Productivity of Software Developers

Group 02: Code4Thought

Ashiph Rai, Victoria Smith Garcia, Benjamin Henderson, Quincy Sproul and Lucia Gallo Garcia.

EMAT30005 Mathematical and Data Modelling

Department of Engineering Mathematics, University of Bristol

1 Introduction

Productivity is a good metric to measure performance in software development teams. Awareness of the level of team productivity is key for the success of a software company [1]. It allows management of companies to better estimate projects' time and cost, to evaluate how their production compares to the market, or when is the right moment to scale up their team. Productivity is therefore an effective approach to control, predict and improve the projects of software companies [2]. A benchmark to measure productivity against coding projects would make managerial decisions easier by giving an estimate of the team's output. In the context of software development there is not a generally accepted definition for productivity [2]. The ISBSG Organisation considers three pertinent factors that affect productivity and which could help to define it: the programming language, the development platform, and the number of active members in the team [3].

Previous literature have studied productivity of developers' teams taking several approaches. In 1977, Walston and Felix [4] measured programming productivity in effort per Source Lines of Code (SLOC). This first major study considered some factors still valid nowadays, such as user participation, constraints on program design or previous programming knowledge. In the same decade, Albrecht [5] studied the effect of the chosen programming language and the project duration with his well-known and widespread function points (FP). In 1987, DeMarco and Lister [6] analysed software teams' productivity from an innovative perspective. They focused on 'soft' factors, such as the project's profit, the work place or time fragmentation. Soft factors are harder to quantify and they are specific to each particular team. This is not viable with a generic productivity model [1]. In the past decades, more advanced research has been carried out using these studies as basis. E.g. Klaus Spiegl [7] or Mashmool et al. [2] who proposed a statistical model to assess a software teams' productivity using agile methods.
In this literature, the majority of the authors focused on the factors that influence productivity. Instead, to create a benchmark, this projects considers productivity to be measured by the output Lines of Code (LOC) per author per week.

We present a benchmark for the productivity of java developers' teams on a weekly basis. To create the benchmark we fit two time series models to our data and find the typical phases in which a project can be split into. The model is valid for back-end projects, typically coded in java, as phases differ for different types of projects. The output is a statistical distribution per week for a project with an expected median of lines per member of the team. The projects used for the benchmark have a broad range of lengths, ranging from 45 weeks to 1184 weeks. One benchmark valid for all projects regardless of their duration, would not give the best productivity estimate. For this reason we present two benchmarks, one for projects of 1184 weeks maximum, and a second one more suitable for shorter projects of up to 520 weeks.

2 Methodology

Git is the largest DevOps tool used for source code management which in 2023 accounted for over 100 million users. [8] Git is well known for its collaborative features and its version control system. This system of repositories keeps a log of all the actions performed and changes occurred at some point in time from an author, the 'commits'. We take the information of these log files by refining the data through a mining process and then transform the log data into JSON files, obtaining 98

in total.

2.1 Data Separation

Project scope, budgetary needs and resource availability are three factors that affect project duration [9]. Long-term projects are usually planned for more extensive and in-depth tasks, and are prone to changes in project specification, where the end product might differ from its initial proposal [10]. In comparison, short-term projects are designed for smaller tasks and require less project and resource management. Due to their dissimilarities, modelling long and short-term projects together can result in a overly generalised model.

Setting the number of weeks for shorter projects to be the same as for the longest project means more data is interpolated that the amount raw data held. This makes the interpolation process less certain, and the interpolated data is less reliable [11]. Consequently, we separate the data into two subsets: long-duration and short-duration projects.

To separate the data, we visualise project duration across our data set in the histogram in Figure 1. Using the Shapiro-Wilk test for normality in Figure 1 gives a p-value of 0.9662 determining that project duration follows a normal distribution with a mean duration of 519 weeks. We use this mean value as a threshold for the separation of subsets. Projects lasting up to and including 519 weeks are categorised as 'short-term projects' and those lasting longer than 519 weeks are labelled as 'long-term projects'. Due to the symmetry of a normal distribution [12], this division results in two equal-sized subsets.

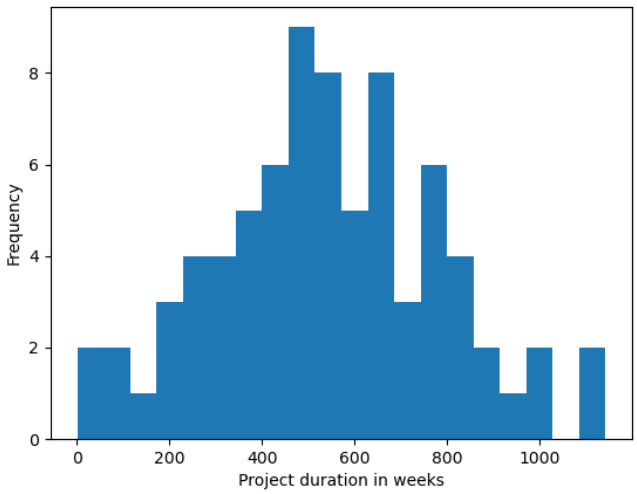


Figure 1: Histogram of project duration in weeks

2.2 Data Pre-processing

Not every commit is pertinent for the benchmark. Back-end and front-end software developers both work on different aspects of a website or software. It would be impractical to expect the workflow

of both to be exactly the same [13]. Instead, this project focuses on developing a productivity benchmark exclusively for back-end software development. We filter the data and retain the projects that contain 'technologies' only, reducing our data to include only java-specific commits.

Renaming variables or function names is a customary process which happens frequently throughout a project's life. Often as the code base grows, the number of variables and functions introduced becomes large and difficult to manage. To improve readability of code, programmers need to choose relevant variable names. This is a rapid and simple action as popular integrated development environments (IDEs) have functions in their systems to effortlessly rename variables. Hence, our model does not consider renaming as a significant commit. It filters out commits that rename variables by deleting commits that include the strings 'Rename', 'Restructure' and '=>' within the subject line of the JSON file.

Commits in which added lines of code equal deleted lines of code, are also filtered out. Small modifications to the code cause Git's version control to interpret the action as removing a line of code and adding a new one. Another consideration is the case of moving code to different sections of the file or to a new file. Again, Git interprets this action as removing and adding lines of code.

To combine the data, it is necessary to stretch the number of weeks of all projects to be the same as the longest project by interpolating the data for each project. Then, with a suitable average measure, we find the average lines per author for each week. The histogram in Figure 5 in Appendix A.1 shows that 78 out of the total 98 projects have between 0 and 1000 lines per author added in their first week. The mean number of lines per author for the first week across all projects is 7205.2 whereas the median is 289. Since the median is a much better representation of this distribution, we use the median lines per author per week to produce our model.

To remove any noise, we use a Savitzky-Golay filter [14]. Peak heights are maintained with this filter, so the general shape of the data is preserved. The filter iterates through windows of data and fits a low order polynomial to data points. To tune this filter, the window length and polynomial order can be altered. The window length should be a value which preserves the shape of the data without distorting the signal tendency. It is desirable to have a low polynomial order as the lower the order, the smoother the filtered data is. We use a polynomial order three and a window length of 2% of the number of weeks. The window length is a percentage of the number of weeks to be representative for the time series it is being used for. In other words, the window length would be larger for a longer time series. We determine these values so that the data sets hold their integrity whilst also being filtered enough to fit a model to. Figure 6 in Appendix A.2 shows a representation of this filter for all projects.

2.3 ARIMA Model

We use historical data to generate a distribution for our benchmarks. In particular, we fit two of the various time series models available: the Autoregressive Integrated Moving Average (ARIMA) model and the Random Walk model.

To choose what time series technique to use, we study the properties of our data. The Augmented Dickey Fuller (ADF) hypothesis test, tests the null hypothesis which questions if a unit root is present in a time series sample, i.e. the data is non-stationary, with a significance level of 0.05. We

obtain a p-value (0.605) greater than 0.05, so the null hypothesis is not rejected. Non-stationary data means that the data has a trend or seasonal pattern, corresponding to the different development phases of a project. The Random Walk model is a non-stationary process so it is suitable for this data. Alternatively, to convert from non-stationary to stationary data, the chosen model needs to include first-order differencing [15]. The ARIMA model is therefore an appropriate model for this data.

The ARIMA model can be adjusted by selecting the right parameters: p, d and q. The parameter 'd' is the number of differences needed for stationarity. To get this value we apply differencing and run two stationarity tests, the ADF test and the Kwiatkowski–Phillips–Schmidt– Shin (KPSS) test until stationarity is achieved. KPSS is a similar hypothesis test to ADF. The null hypothesis (for KPSS test), a unit root is not present in a time series sample, i.e. the data is stationary, is tested against a significance level of 0.05. We obtain a p-value (5.256×10^{-18}) smaller than 0.05 and the null hypothesis is not rejected, so data is stationary after one differencing. The purpose of running two tests is to make sure both agree on the result. Both tests determine the pre-processed data is non-stationary and after one differencing, it becomes stationary. Hence, the degree of differencing is one. Parameters 'p' and 'q' refer to the autoregressive (AR) and the moving average (MA) terms, respectively. They are the number of past points considered to predict the next point (p), and the number of past prediction errors observed to predict future points (q) [16]. They can be obtained through the Partial Autocorrelation function (PACF) plot, top plot of Figure 2, for the number of AR terms and the Autocorrelation function (ACF) plot, bottom plot of Figure 2, for the number of MA terms.

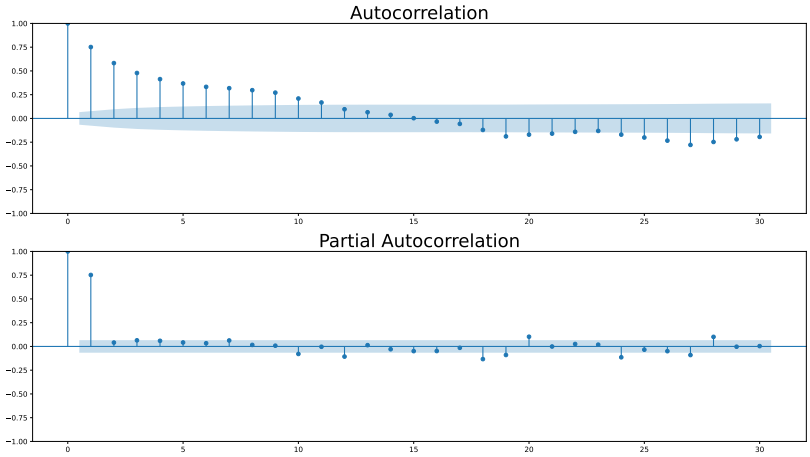


Figure 2: Autocorrelation function plot (top subplot) and Partial Autocorrelation function plot (bottom subplot) after differencing the data once.

In the PACF plot, there are two spikes above the significance region (blue shaded region). The first one shows a correlation of one to itself so it is not considered. Lag 1 is significant as it is well above the significance region, so we set $p = 1$. Looking now at the ACF plot, starting from lag 1

there are various spikes above the significance. This time, we consider the three first ones to be significant as a simpler model will avoid over-fitting, so set $q = 3$. The final model can be written as: $ARIMA(p = 1, d = 1, q = 3)$.

2.4 Data-Driven Random Walk

The Random Walk time series simulates how the productivity evolves over time by taking into account past performance. It considers whether the number of lines per author for each week increases $(+1)$ or decreases (-1) compared to the previous week's value. These values are stored and summed up with the absolute value of the increase or decrease in lines per author. Doing this gives us an overall picture of the direction and magnitude of changes in the number of lines per author over time.

In a standard Random Walk model, the step size and the probability of moving in either direction is fixed and unbiased. However, our data-driven Random Walk uses a different probability value each week. We find the probability of observing an increment or a reduction by dividing the sum of the directions by the sum of the magnitudes. The values obtained are scaled between 0 and 1 so that a probability of 0 and a probability of 1 indicate a 100% chance of a decrease or increase in lines per author respectively. Our Random Walk calculates the step size for each iteration as the difference in number of lines of code per author between each week and the previous one.

2.5 Phase Separation and Change Point Analysis

We assume the projects have three phases: initial, midterm and mature; and we split the benchmarks accordingly. Productivity of a team at a certain phase can be compared against the same phase in the benchmark.

Change point analysis (CPA) performs a searching algorithm to find 'changes' in a time-series model and segments it into phases. CPA can only be performed on stationary data, i.e. after one differencing of our pre-processed data. There are two types of CPA: 'Online' CPA, used to detect anomalies on live data, and 'offline' CPA, that focuses on the segmentation of data already collected. Offline CPA is more adequate for our purpose since our data is collected and already processed.

We opt for a binary segmentation method. It is an iterative algorithm that finds first one change point in the signal and splits the signal at it. Then, the algorithm is performed again on these two resulting sub-signals, and so on up to N change points. We select two change points to symbolise the change from the initial to the midterm phase and from the midterm to the mature phase. To find the two change points, we use the kernelised mean change model (KMC) which finds change points based off changes in the mean of the embedded signal. In this case, we use the radial basis function (RBF) as the kernel of the KMC model due to its simplicity.

3 Results

We propose four benchmarks for the expected productivity of developers considering the phase of their project. Figure 3a is the ARIMA benchmark for projects of duration up to 520 weeks,

and Figure 4a is the ARIMA benchmark for longer projects, up to 1184 weeks. Similarly, Figure 3b is the Random Walk benchmark for short-term projects, and Figure 4b is the Random Walk benchmark for long-term projects.

After fitting an ARIMA model, it is a good point of analysis to determine if the lag variables of the model are significant. In our models, there are four lag variables: one for the autoregression (AR) lags and three for the moving average lags (MA). A lag variable should have an associated p-value smaller than a significance level of 0.05 to deem it statistically significant. In the ARIMA model for short-term projects, two moving average parameters have p-values of 0.872 and 0.332, both larger than 0.05. This suggests they are not statistically significant and could be ignored. The p-values for the lag variables in the long-term benchmark are all less than 1×10^{-4}, hence smaller than 0.05 deeming them all statistically significant.

Following with the analysis, we check if the auto-correlations in the residuals are non-zero. We do this by using a Ljung-Box test with a significance level of 0.05. The null hypothesis is that the data is independently distributed. The test statistics (Q) for the short-term and long-term benchmarks are 0.85 and 0.77 respectively, both greater than the significance level. Thus, the null hypothesis is rejected, meaning the data is serially uncorrelated and that the benchmarks are a good fit for their respective data sets.

To further validate the ARIMA models, we calculate the Mean Absolute Percentage Error (MAPE). MAPE is a commonly used method for validation. Accuracy is determined by $100 - MAPE$, we write MAPE as

$$MAPE = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{A_i - P_i}{A_i} \right| \times 100 \tag{1}$$

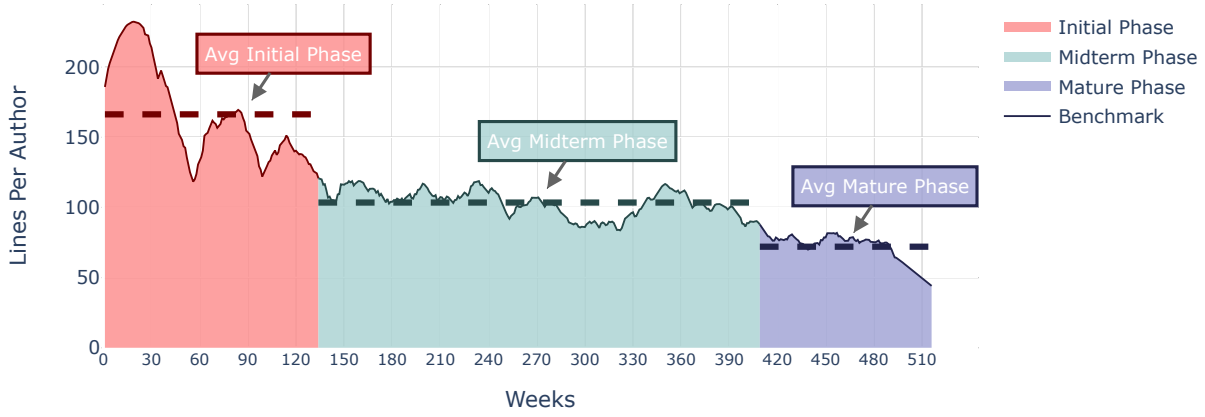
where n is the number of times the summation iteration happens, P_i are the predicted values from the model and A_i are the actual values from the empirical data. For short-term projects $MAPE = 0.9\%$. The accuracy is then calculated as $100 - MAPE = 100 - 0.9 = 99.1\%$. For long-term projects $MAPE = 0.7\%$ and has an accuracy of $100 - 0.7 = 99.3\%$.

To validate the Random Walk models, we fit the time series over 100,000 times, each time with a new set of random variables. We then calculate the MAPE between the model obtained and the original distribution for each iteration. The best Random Walk model is chosen to be the one with the lowest MAPE as it is the best fitted to the distribution. For short-term projects $MAPE = 7.4\%$ and its corresponding accuracy is 92.6%. For long-term projects $MAPE = 15.2\%$ and the accuracy is $100 - 15.2 = 84.8\%$.

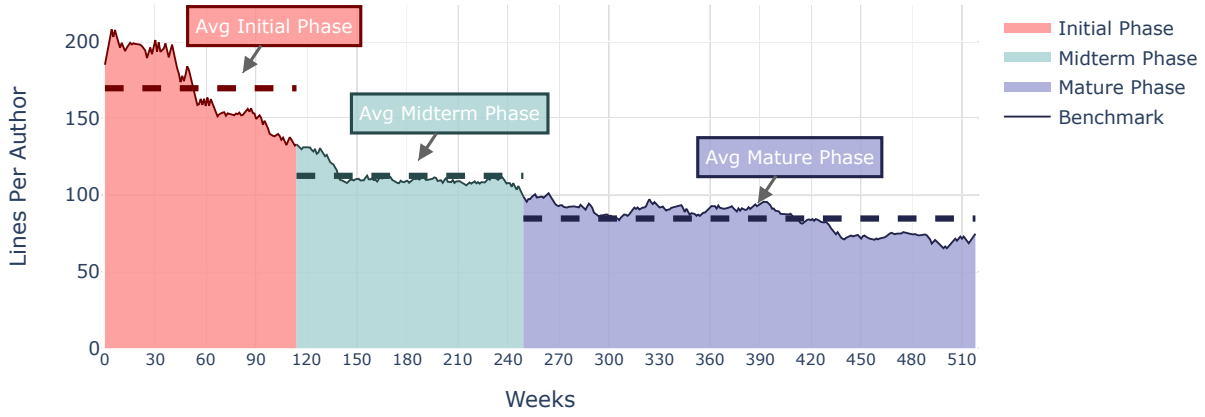
3.1 Comparison of the models

The benchmarks show an overall decreasing trend. This is in accordance with the nature of programming projects, in which most of the code is written during the initial and midterm phases. We notice that the ARIMA models present abrupt spikes compared to the smoothness of the Random Walk models. This is due to the properties of the ARIMA time series, as ARIMA tends to fit very well to the distribution. In Section 2.2, we see how the Savitzky-Golay filter cannot completely

remove noise as doing so would lead to the loss of important information. Hence, the distribution has some spikes which are still noticeable after fitting the ARIMA time series.

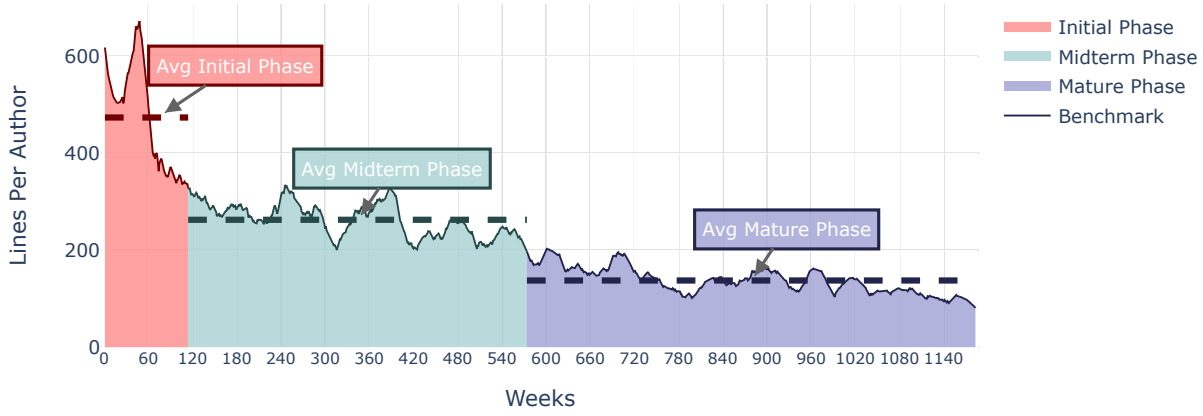


(a) Short-term ARIMA benchmark

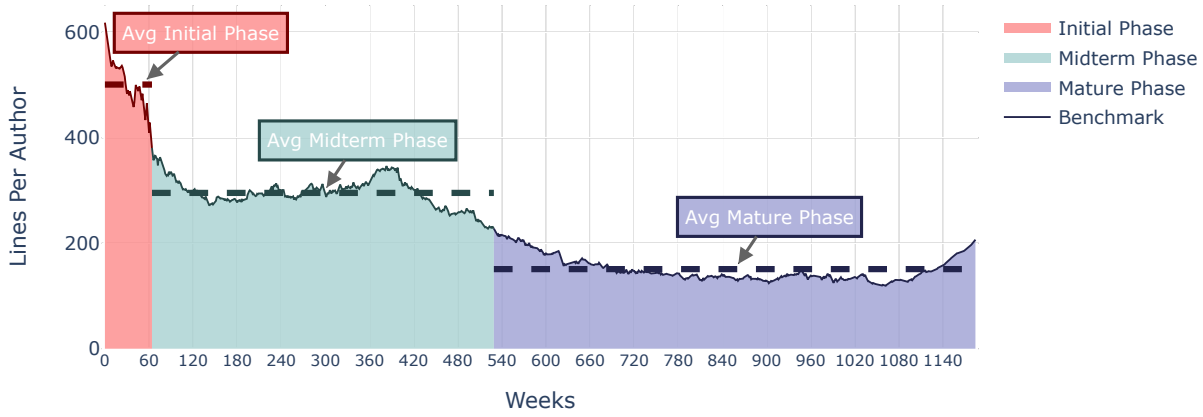


(b) Short-term Random Walk benchmark

Figure 3: Benchmarks for short-term projects. Both figures show the different phases. Also show the average lines per author for each phase (values can be found in Table 1).



(a) Long-term ARIMA Benchmark



(b) Long-term Random Walk benchmark

Figure 4: Benchmarks for long-term projects. Both figures show the different phases. Also shows the average lines per author for each phase (values can be found in Table 2).

By looking at the short-term and long-term benchmarks for the same models, we see clear dissimilarities in the phases' duration and overall trend. This confirms that separating the initial data set into two subsets and creating different benchmarks for them was a good decision.

Figure 3 shows the biggest difference between both short-term models is the duration of the midterm and mature phases. This can be caused by the different levels of noise in the data, which is captured better by the ARIMA model. This noise can distort the change point analysis and lead to different phase boundaries being found. While both models present an initial phase of similar duration, the midterm phase in the ARIMA model lasts 275 weeks, twice as long as in the Random Walk model, which lasts 135 weeks. Contrariwise, the mature phase in the ARIMA model only lasts 110 weeks compared to the 270 weeks in the Random Walk model. Following with the comparison, Table 1 shows the similarity in the benchmarks for short-term projects. Across the three phases, the models have an average absolute difference of 8.3 lines of code per author.

Model	Initial Phase	Midterm Phase	Mature Phase
ARIMA	166.3	103.5	72.0
Random Walk	169.5	112.5	84.6

Table 1: Average lines per author for each phase in short-term projects from the ARIMA and Random Walk models

In Figure 4, we observe how for long-term projects, the ARIMA and the Random Walk models both output similar phase lengths. The main discrepancy between them lies in the duration of the initial phase. In the ARIMA model the initial phase lasts 120 weeks, twice as long as in the Random Walk. The average benchmark values for long-term projects, shown in Table 2, have an average absolute difference of 24.2 lines of code per author across all phases, between both models. This value is much higher than that for short-term projects.

Model	Initial Phase	Midterm Phase	Mature Phase
ARIMA	474.0	263.0	137.7
Random Walk	501.0	295.4	151.0

Table 2: Average lines per author for each phase in long-term projects from the ARIMA and Random Walk models

Overall, both models perform alike, with close average benchmark values and similar downturn trends. The Random Walk models appear to be harsher for both short-term and long-term projects as the average benchmark values for each phase for both benchmarks, are greater than those of the ARIMA models. These Random Walk models establish that a productive team should be committing more lines per author per phase than the ARIMA models.

3.2 Testing

The testing set consists of three short-term projects and three long-term projects. Their mean lines per phase are computed for each phase duration produced by the ARIMA and Random Walk models.

Model	Test Project	Initial Phase	Midterm Phase	Mature Phase
ARIMA	1	858.0	251.7	473.3
	2	422.5	481.1	210.6
	3	268.2	76.9	915.0
Random Walk	1	956.3	244.0	322.0
	2	433.7	634.6	306.7
	3	307.3	74.9	79.5

Table 3: For short projects: Average lines per phase compared against the ARIMA and Random Walk benchmark values from Table 1. Green and red cells show values above and below the benchmark, respectively.

Table 3 shows that the ARIMA and Random Walk models give mostly similar results for the three projects. However, the mature phases of the two short-term benchmarks are very different in length, potentially resulting in disagreement between predictions by the two models in the mature phase. See test project 3; in the mature phase, it is above the ARIMA benchmark but below the Random Walk benchmark.

Model	Test Project	Initial Phase	Midterm Phase	Mature Phase
ARIMA	1	464.3	208.1	208.9
	2	1392.3	265.0	102.4
	3	2114.4	2152.6	2522.2
Random Walk	1	536.2	222.1	216.6
	2	1716.5	355.5	104.3
	3	1131.7	2020.9	2685.0

Table 4: For long projects: Average lines per phase compared against the ARIMA and Random Walk benchmark values from Table 2. Green and red cells show values above and below the benchmark respectively.

Table 4 shows again similar results for three long-term test projects. For test project 1, it can be seen that the ARIMA benchmark predicts the project to be under-performing in the initial phase whereas the Random Walk predicts the project is over-performing. These differences are a result of the different phase lengths, as previously discussed.

4 Discussion

Our models were able to provide a benchmark for the productivity of java developers' teams, see Figures 3 and 4. Despite some of the projects having authors contributing with less than ten lines of code for the entire project duration, see Figure 7 in Appendix A.3, we considered everyone in the team to be a contributor. These authors' minimal input is not representative so their consideration can significantly skew the benchmark. For this reason, they could be considered as 'inactive' and filtered out by setting a normalised threshold on the minimum number of lines committed for the author to be considered 'active'.

When tuning the ARIMA model, we determined the parameters p, d and q to best suit our entire data set, before the split into two subsets. We then used them to fit an ARIMA time series to the short and long projects distributions. Our ARIMA models could be improved if instead we determine the appropriate parameters for both subsets separately. This is evident from our findings in Section 3 where we found that two lag variables may be statistically insignificant in the model for short-term projects yet statistically significant in the model for long-term projects. This suggests that a simpler ARIMA model, which does not include these lag variables may be a better fit to the short-term project data.

All of our models presented in this report are reproducible with similar projects. They can be replicated and applied to any other programming technology or project duration that has data available in GitHub. The benchmarks created are valid for back-end programming projects, specifically those coded in Javascript. Although Javascript has been considered the leading programming language for back-end developers for 10 years in a row [17], other languages such as Python, PHP and C# are still very popular, with 48% of back-end developers using Python [17]. As an extension to this paper, a benchmark for projects in other programming languages could be produced.

We divided the models into three phases. This assumption can be a limitation for projects with a different number of phases. Further analysis could be done by using CPA without inputting the number of phases. This would require us to modulate a 'penalty' parameter which is hyper sensitive. This may lead to CPA calculating more realistic phase boundaries.

Our models are heavily reliant on the collected data. Within our collection of projects, we do not know how productive each project was. This offers an interesting opportunity for the model to fit on biased data. Intentionally adding bias to the data will lead to a specific benchmark with a desired level of effort i.e choosing only successful projects would lead to a benchmark for high-achieving projects. This also results in an uncertainty in testing. As we are unsure on the productivity of our testing projects, we cannot determine whether the models accurately predict productivity.

5 Conclusion

This technical report proposes four benchmarks for the productivity of java developers' teams on a weekly basis using 98 projects sourced from GitHub. For the projects, productivity is measured by the number of lines of code committed per week to a project's GitHub repository. An ARIMA and a Random Walk time series models were fitted to a distribution of short-term projects and one of long-term projects. For short-term projects, the ARIMA and the Random Walk benchmarks obtained a training accuracy of 99.1% and 92.6%, respectively. For long-term projects, the ARIMA benchmark obtained a training accuracy of 99.3% and the Random Walk benchmark of 84.8%. Both models produce similar benchmarks, so either could be used by developers' teams to measure their productivity levels. The main difference lays in the phases' duration. Developers can choose the ARIMA or the Random Walk benchmark depending on which method has a more similar phase structure to their projects. Moreover, when deciding which model to use, developers could consider how the Random Walk model presents a harsher benchmark and whether this is more or less related to their project.

The benchmarks are highly dependent on the 98 projects used in the training set. However, the

model's ability to generalise can be improved by replicating the models using other programming languages, different projects duration, other number of phases or a larger training set.

References

[1] S. Wagner and M. Ruhe, "A systematic review of productivity factors in software development," 2018.

[2] A. Mashmool, S. Khosravi, J. H. Joloudari, I. Inayat, T. J. Gandomani, and A. Mosavi, "A statistical model to assess the team's productivity in agile software teams," in *2021 IEEE 4th International Conference and Workshop Óbuda on Electrical and Power Engineering (CANDO-EPE)*, pp. 11–18, 2021.

[3] I. S. B. S. G. (ISBSG), "Team size impacts special report, technical report," 2007.

[4] C. E. Walston and C. P. Felix, "A method of programming measurement and estimation," *IBM Systems Journal*, vol. 16, no. 1, pp. 54–73, 1977.

[5] A. Albrecht, "Measuring application development productivity," *Proceedings of IBM Applications Development Symposium*, pp. 83–92, 1979.

[6] T. DeMarco and T. Lister, *Peopleware. Productive Projects and Teams.* Dorset House Publishing, 1987.

[7] K. Spiegl, "Projektmanagement life - best practices und significant events im software projektmanagement," Master's thesis, January 2007.

[8] Wikipedia, "Github." `https://en.wikipedia.org/wiki/GitHub`, 2023.

[9] C. Contributor, "Differences between short-term & long-term projects." `https://work.chron.com/differences-between-shortterm-longterm-projects-11637.html`, 2021.

[10] A. Fedin, "What is project management in long-term projects?." `https://medium.com/111-minutes/what-is-project-management-in-long-term-projects-30a937bf39ce`, 2015.

[11] C. Labs, "The dangers of data interpolation and its affect on data historians." `https://blog.canarylabs.com/the-dangers-of-data-interpolation-and-its-affect-on-data-historians`, 2020.

[12] C. Team, "Normal distribution." `https://corporatefinanceinstitute.com/resources/data-science/normal-distribution/`, 2022.

[13] K. A. S. N. H. University, "Front end vs backend: What's the difference?." `https://kenzie.snhu.edu/blog/front-end-vs-back-end-whats-the-difference/`, 2022.

[14] A. Savitzky and M. J. E. Golay, "Smoothing and differentiation of data by simplified least squares procedures.," *Analytical Chemistry*, vol. 36, no. 8, pp. 1627–1639, 1964.

[15] C. One, "Understanding arima models for machine learning." `https://www.capitalone.com/tech/machine-learning/understanding-arima-models/`. Last accessed: 2023-02-01.

[16] E. Heckman, "Fitting an arima model." https://blog.minitab.com/en/starting-out-with-statistical-software/fitting-an-arima-model, 2016. Last accessed: 2023-02-05.

[17] N. T. PROFESSIONALS, "Top 7 backend programming languages in 2022." https://nexttechnology.io/top-7-backend-programming-languages-in-2022/, 2022.

[18] C. Truong, L. Oudre, and N. Vayatis, "Selective review of offline change point detection methods," *Signal Processing*, vol. 167, p. 107299, 2020.

[19] Wikipedia, "Shapiro-wilk test." https://en.wikipedia.org/wiki/Shapiro%E2%80%93Wilk_test.

A Other materials

A.1 Appendix A

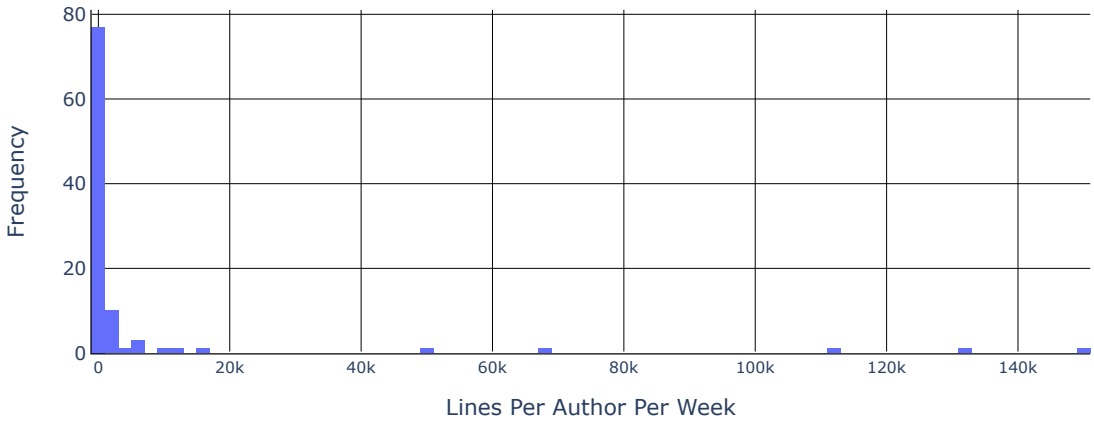


Figure 5: Histogram showing the distribution of the number of lines per author in the first week for all projects.

A.2 Appendix B

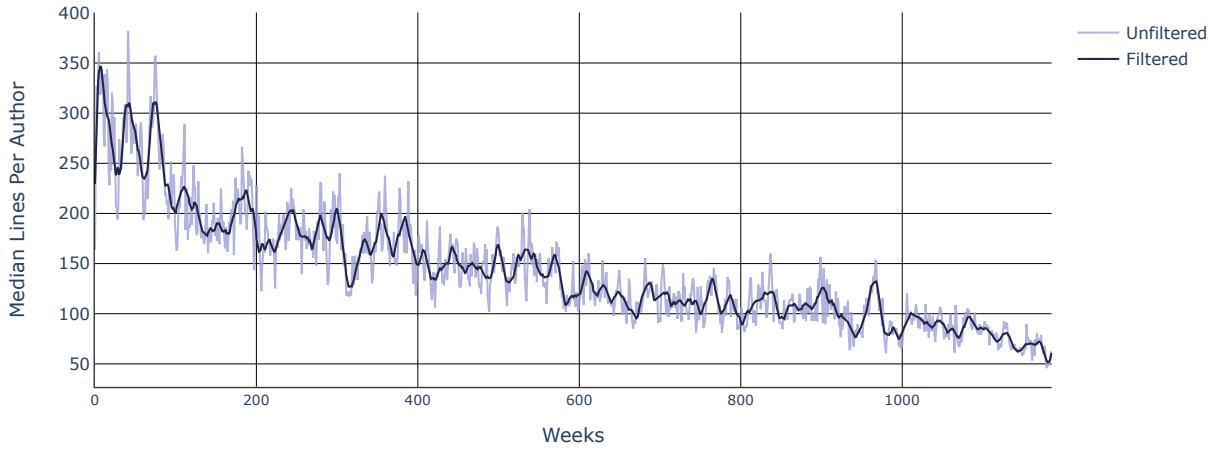
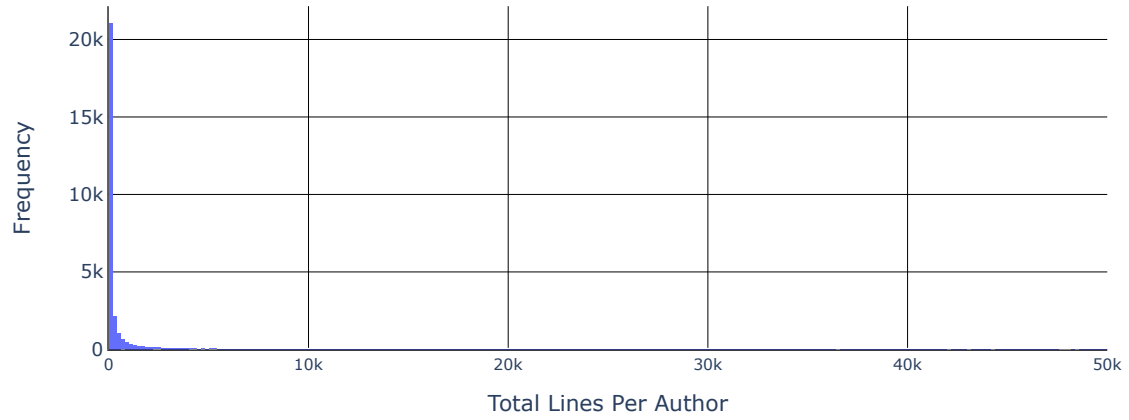
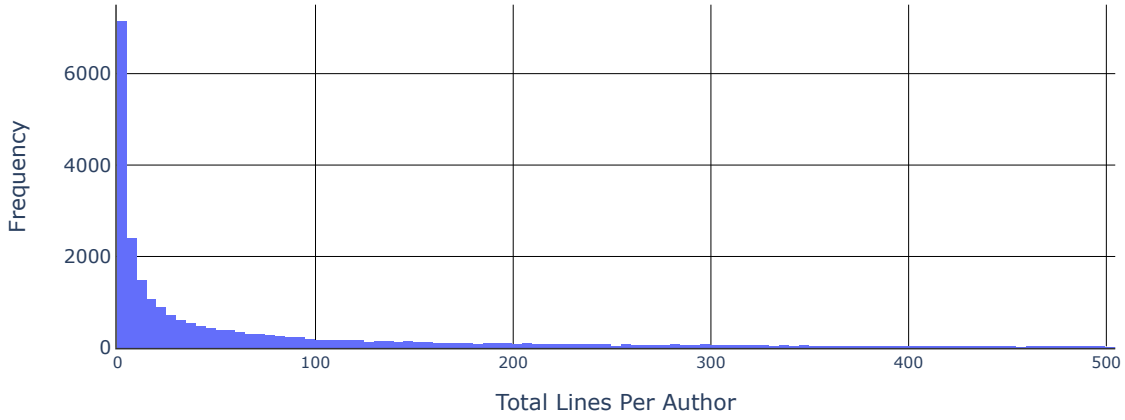


Figure 6: Lines per author for all projects. Blue is unfiltered (raw) data and orange is filtered data using the Savitzky-Golay filter with polynomial order 3 and window size 2% of number of weeks.

A.3 Appendix C



(a) Total lines per author values between 0 and 50,000.



(b) Enlarged version of 7a for total lines per author values between 0 and 500.

Figure 7: Histogram of the total lines of code committed per author for the entire project

A.4 Appendix D: GitHub Repository

The GitHub repository containing the code used to produce the results in this paper can be found at:

`https://github.com/lj20127/MDM3_Code4Thought.git`