The Evolution of Software Entropy in Open Source projects: An Empirical Study

Umme Ayda Mannan
Oregon State University
Corvallis, OR
mannanu@oregonstate.edu

Iftekhar Ahmed
University of California, Irvine
Irvine, CA
iftekha@uci.edu

Carlos Jensen
University of California, San Diego
San Diego, CA
cjensen@ucsd.edu

Anita Sarma
Oregon State University
Corvallis, OR
anita.sarma@oregonstate.edu

ABSTRACT

Software entropy impacts the overall quality of software systems. High entropy hinders developers from understanding the purpose of a piece of code and can cause developers to make sub-optimal changes and introduce bugs. Researchers have used entropy scores to measure the naturalness of code. However, thus far, no one has investigated how entropy evolves in real-life software over time and how that relates to software evolution. We perform a large scale study of 158 open-source Java projects and find that in general, entropy shows an increasing trend among the open-source projects. We also find that on average week wise entropy of a project has a moderate correlation with project metrics (number of active contributors per week, total methods per week, and total files per week) used to investigate project evolution. Our findings contribute to understanding the evolution of software entropy, which hint towards the need for tools to provide real-time entropy scores to the developers to help them organize the code.

CCS CONCEPTS

• **Software and its engineering** → *Software reliability*; *Software maintenance tools*; *Software design tradeoffs*.

ACM Reference Format:
Umme Ayda Mannan, Iftekhar Ahmed, Carlos Jensen, and Anita Sarma. 2020. The Evolution of Software Entropy in Open Source projects: An Empirical Study. In *Proceedings of* . ACM, New York, NY, USA, 7 pages.

1 INTRODUCTION

During the lifetime, software systems change for various reasons, such as adding new features, fixing bugs, or refactoring. Due to time constraints, limited resources, and the lack of regular maintenance, these changes could deteriorate the structure of the software system from its original state, thus increase the source code complexity and, in general, make the system more challenging to understand and maintain in the future. Researchers have used entropy to quantify complexity and disorganization [6, 14]. As high entropy makes the code difficult to understand, it makes the code more bug-prone.

Hellendoorn et al. [15] observed that code quality is also associated with how developers write code. Programmers tend to be highly repetitive and predictable while developing real-world software [12]. Hindle et al. were the first to capture the repetitiveness in a statistical language model [16]. They called this property the naturalness of code and measured it by entropy. If a code snippet has high entropy, that means the code is drifting from its natural state. Previous research shows that high entropic code is associated with future bugs [24] and design degradation [10]. Ray et al. [24] investigated if there is any correlation between the buggy code and entropy. They observed that buggy codes are, in general, less natural, i.e., they have higher entropy than non-buggy code. Chatzigeorgiou et al. showed that higher entropy also impacts the design quality of a software system [10].

Previous research has investigated the impact of entropy on design quality as well as code quality. However, the overall situation of software entropy in real-life software has not been investigated yet. This study aims to shed light on how entropy evolves as software ages and the relationship between entropy evolution and project evolution in terms of the number of contributors and number of methods and files. More specifically, we try to answer the following research questions:

RQ1: How entropy evolves over time in OSS projects?

RQ2: What is the correlation between entropy and project evolution in terms of project size and the number of contributors?

To answer our research questions, we conducted a large scale empirical study. We sampled 158 projects from GitHub [13] and calculated the week wise entropy for each project to investigate the evolution of entropy across OSS projects over 330 weeks. We also collected project metrics for each week for each project. Furthermore, we investigate the association between entropy and other project metrics by performing a cross-correlation as all of our data is time-series data.

The paper is structured as follows, Section 2 presents our methodology, the demographics of our corpus, data collection, and analysis process. In Section 3, we present our findings. Section 4 discusses the results and outlines implications for developers and researchers. Section 6 provide a review of prior research efforts. Section 7 concludes with a summary of the key findings and future work.

2 METHODOLOGY

The goal of this empirical study is to understand how entropy in OSS projects evolves and the relationship between entropy evolution and project evolution. For this study, we create a corpus of 158 open source projects. In the following subsections, we describe the pipeline we followed to collect, process, and analyze the data.

2.1 Project Selection Criteria

To make sure our findings would be representative of the code developed in the real world, we selected active OSS Java projects from GitHub. We selected Java projects because Java is one of the most popular languages [27]. We adopted the initial project selection criteria from the study by Ahmed et al. [5]– searching and selecting initial project, project size, and the number of files in a project. Then we added additional criteria as discussed next for project selection.

We start by randomly selecting 900 projects by searching for java projects in the GitHub search engine. Next, to make sure the projects in our corpus is representative of real-world projects and not throw-away or class projects, we followed the guidelines by Kalliamvakou et al. [19]. Furthermore, following the guidelines, we eliminated the repositories that are not "actual" projects, have very few commits, are inactive, are not related to software development, or are personal projects. This left us with 500 projects. Then we selected only the projects that meet the following criteria,

- Project size (Must have more than ten files and 500 lines of code).
- Project age (More than ten weeks).
- Commit history (More than ten commits).

This resulted in our corpus containing 158 projects. Table 1 provides a summary of our selected projects.

Table 1: Project Statistics

Dimension	Max	Min	Average	Median	Std dev
Line count	804,658	686	34,582.51	11,188	83,536.64
File count	1,693	11	165.4845	87	253.5198
Total Commits	6,256	11	485.2981	183	847.7926
# Developers	157	1	19.37267	9	26.67907
Duration (weeks)	782	10	244	235	168.1799

2.1.1 Unit of measure selection. To investigate how entropy in OSS projects evolve, we could use different ways of partitioning time. Izurieta et al. [17, 18] have used releases as the unit of time, where others have used individual commits, or discrete-time units (years, months, weeks, days) [5, 21]. As individual commits would be too fine-grained for our purpose, we selected the week as our unit of measure similar to Ahmed et al. [5]. Because it gives us enough detailed insight into the evolution of projects. We then checked the distribution of commits across the history of the projects and found that the majority (90%) of the projects had an active history of 330 weeks or less. We cut off our analysis at 330 weeks in order to prevent extremely long-lived projects from skewing the results.

2.2 Measuring Entropy

For our study we calculate week wise total entropy for each project. To get the total entropy, first we calculate the entropy for each java source file using Python's `Entropy` library [1]. The library uses Shannon's entropy [2]. Other researchers also used Shannon's entropy [8] $H(X)$, which is calculated using the following formula:

$$H(X) = \sum_{i=1}^{n} P(x_i) \log_2 P(x_i) \quad (1)$$

In Equation 1, X is a file, in our case java source file. And x_i is a term (in our case a token in a java method) in X. $P(x_i)$ is the probability of a change occurring in a file. n is the total number of methods in each java source files. We will get a maximum entropy when all methods have the same probability of having a change: $P(x_i) = 1/, \forall i = 1, 2, \ldots, n$. We will get minimum entropy if for a method i, $P(x_i) = 1$, and for all other methods, other than i, $P(x_j) = 0, \forall j \neq i$.

After collecting entropy for each java file for each week, we sum up the entropy scores of all java files in a project to get week wise total entropy for that project.

2.3 Collecting Project Metrics

To answer our second research question, we collected different project related metrics to investigate the correlation between project entropy and project evolution. Table 2 lists the metrics we collected for this purpose.

Table 2: Project metrics used for investigating project's evolution.

Metric	Description
Total methods per week	Week wise number of unique methods for each project.
Total contributors per week	Week wise number of unique contributors for each project.
Total files per week	Week wise number of unique files for each projects

After collecting total entropy for each project, we collected the above-mentioned project metrics (Table 2) from our data set. We use Understand [3] to collect the number of files and number of methods for each week, for each project. We collect the number of unique contributors per week from the project repository by counting the number of developers who made commits in the codebase in a week.

2.4 Data Analysis

We calculated the total entropy and other project metrics (mentioned in table 2) for each project over 330 weeks. To compare these time series, first, we normalized the data since the number of metrics will vary according to the size of the development team and project. There are many ways of normalization, and the most commonly used one is dividing the data by the lines of code. Since the aim of or study is to identify general trends across projects, not to look at differences between them, we normalized all the week wise collected metrics using the feature scaling [7], which gives a score between 0 and 1 (Equation 2).

$$\text{Rescaled value} = \frac{x - min(x)}{max(x) - min(x)} \quad (2)$$

Where,
x = each data point.
min(x) = The minimum among all the data points.
max(x) = The maximum among all the data points.

One of our goals is to identify the correlation between total entropy per week and the other project-related metrics per week. We calculate the cross-correlations between entropy per week, and total methods per week, total contributors per week and total files per week individually for each project [22]. As the first step of time series analysis, we start by checking if there are any visible trends. If a time series exhibits a visible trend, we need to remove the trend before further analysis. This process is called detrending. We applied the first differencing method to detrend time series [22], and after that, we calculate the cross-correlation.

3 RESULTS

In the following sections, We structure our study findings based on our research questions.

3.1 How entropy evolve over time

To answer our first research question, we start by looking into the general trend of average entropy across all projects. Figure 1 shows the *increasing* trend of the average entropy across all projects.

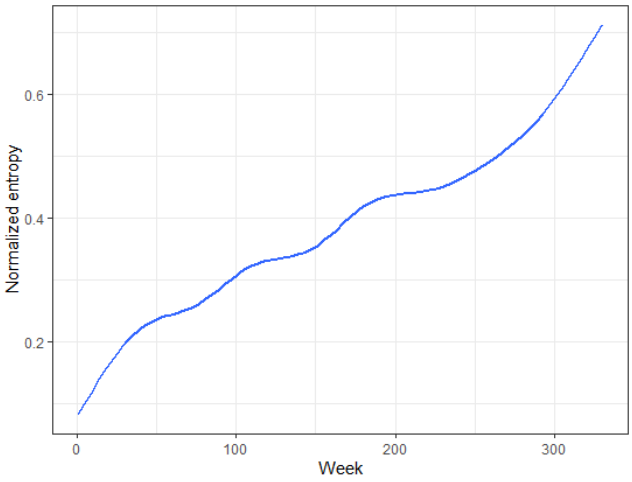

Figure 1: Week-wise average entropy for the projects.

We also wanted to check whether this same trend holds for all projects. When we looked into each of the projects individually, four different trends emerged. The four categories are:

Cat 1: Projects start with low entropy and increase over time.
Cat 2: Projects start with low entropy and remain low over time.
Cat 3: Projects start with high entropy and increase even more over time.
Cat 4: Projects start with high entropy and decrease over time.

Figure 2 presents examples of these trends using specific projects from our corpus. Category 1 includes projects that start with low entropy and increase as the project matures, manifests in “artifactory-client-java" project. Category 2 includes the projects that have a constant low entropy trend over the time; “jitescript" is an exemplar of this trend. Category 3 includes projects that start with high entropy and increase over time, as seen in the project “gelfj". Furthermore, category 4 includes the project that starts with a high entropy but decreases over time, as seen in the project “jmimemagic". Table 3 shows the percentage of projects in each entropy trend category in our corpus. We also check if these trends are statistically significant or not and found that 76% of all projects have statistically significant trends.

Table 3: Number of projects in each entropy trend category

General Trend	Number of projects(%)
Cat 1	69.62%
Cat 2	2.53%
Cat 3	15.82%
Cat 4	12.03%

Observation 1: Different types of entropy trends exist, however on an average entropy increases over time.

3.2 Association between entropy and other project metrics

To answer our second research question, we investigate the correlation between entropy and other project metrics. As entropy is related to source code, we picked the project metrics that are also related to source code, total methods, total files, and total contributors.

We start by looking into the general trend of the average number of methods, files, and contributors across all projects. Figure 3 shows a *increasing* trend for each of the project metrics across all projects.

Since entropy per week, methods per week, contributors per week, and total files per week all are time series data; a time series analysis is required to identify the correlation between them, which is called cross-correlation. We do time series analysis between entropy and each metrics(mentioned in table 2.3) individually for each project. Time series analysis requires a pre-processing step before doing the actual correlation analysis [22]. Due to space limitations, we report the results after each pre-processing step in the companion website[20].

Next, we calculate the correlation between each pair of the following time series data, entropy and total methods, entropy and total developers and entropy, and total files.

Figure 4 shows the result of this step for each pair as an example. From the figure, we see that the highest correlation value between entropy and method for “argparse4j" project is 0.58 (shown by the circled vertical line in (a) at a lag of 7. Similarly, (b) in figure 4 shows the correlation between entropy and developer with the highest correlation value of 0.56 for the project “clj-ds" and (c) shows and the correlation between entropy and total files for project “cloudhopper-smpp" with the highest correlation value of 0.47 at lag 6. Table 4 shows the statistics of the cross-correlation values with the average lags for each pair of time series across 158 projects.

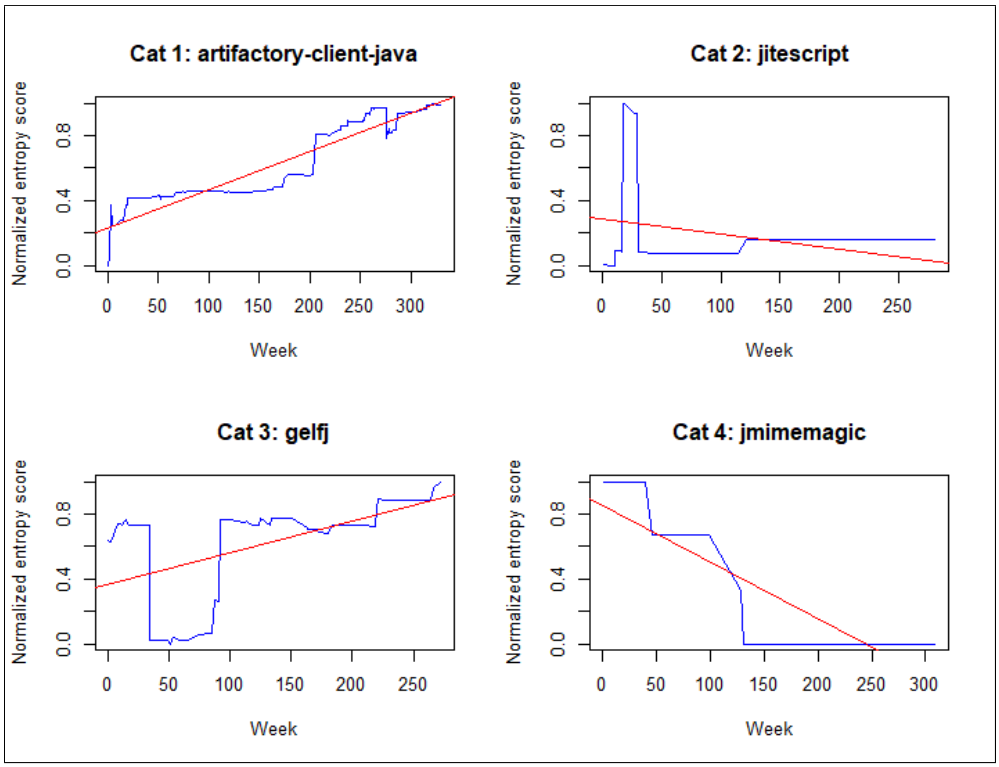

Figure 2: Week wise project's entropy trend.

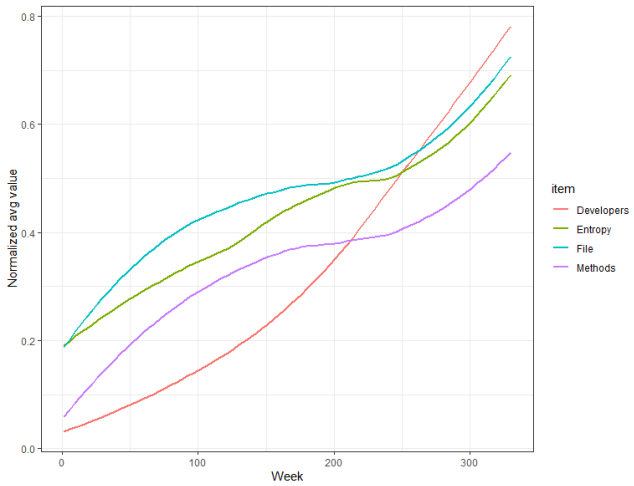

Figure 3: Week-wise average for each metrics across all projects.

Next, we calculate the significance of the correlation for each project for all three pairs. A correlation is significant when the absolute value is greater than, $\frac{2}{\sqrt{n-|k|}}$, where n is the number of observations and k is the lag [4]. We found that 150 out of 158 projects have ave statistically significant cross-correlation between entropy and number of methods. For the pair entropy and developers, we found that 135 projects have a statistically significant correlation, and between entropy and total files, 143 projects have a statistically significant correlation. We also looked into the lags between the pair of time series. The fourth column of the table 4 shows the average lags for each paper of time series. Due to space limitations, we report the plots for each paper, for each of the 158 projects in the companion website [20].

Table 4: Distribution of the correlation value of each pair of time series.

	Maximum	Minimum	Median	Avg lags
Entropy and Methods	0.94	0.067	0.45	7.48
Entropy and Developers	0.94	0.082	0.36	10.51
Entropy and Files	0.96	0.07	0.35	7.5

Observation 2: Entropy is moderately correlated with other project metrics.

4 DISCUSSION

From our analysis, we found that the overall entropy scores increase over the life of OSS projects (Figure 1). One of the reasons behind this increasing trend could be the practice of software developers implementing quick fixes with immature design solutions instead of the optimal solution due to the time constraints. Over time, these bad design solutions build-up, and software design degrades,

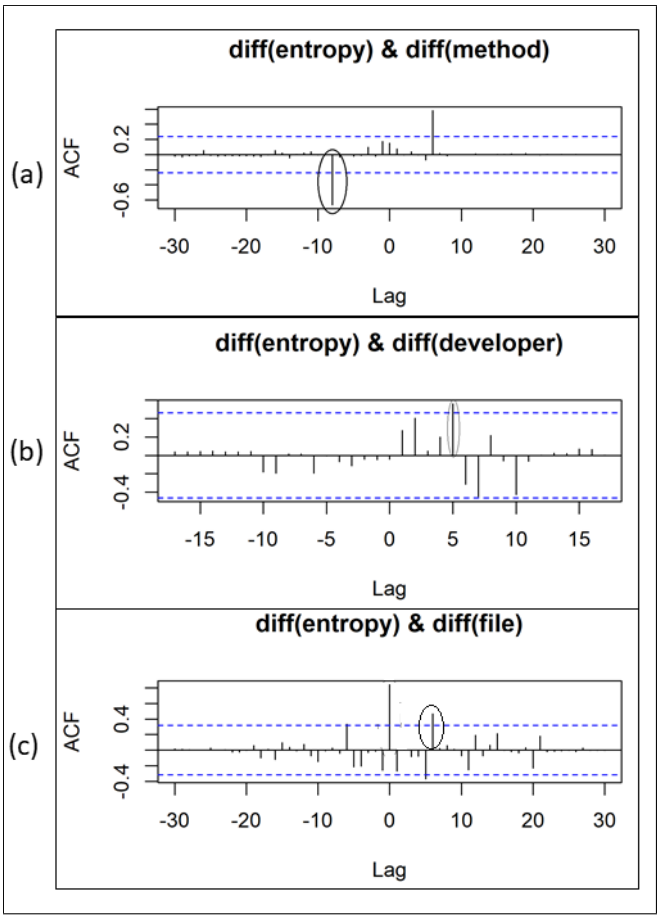

Figure 4: Cross correlation values of two time series

and entropy (which is a measurement of software degradation [6]) increases.

However, when we investigate the entropy trend for each of the projects over 330 weeks, we found four distinct categories of trends, as mentioned in section 3.1. From our analysis, we found that 76% of the projects' trends are statistically significant. Among these four categories, category 2 and category 4 was showing a low and decrease entropy respectively over time. Though these categories have a small number of projects (See table 3), we decided to manually check some of them to find out if there is any specific practice that helping these projects to maintain or decrease the entropy of their project. We selected one project from category 2 and 3 projects from category 4 and went through the GitHub pages for these projects. For all five projects, we noticed that the instructions for contributors are well defined. These projects are not only provided instructions about using the software but also provided specific information to contributors regarding the structure of the codebase, which package contains important classes, preferred development tools, testing instructions on how to format the code, specific coding style, etc. These projects also have specific instructions on where to report a bug, if they are using any issue tracking system or not. One of the projects named "mapdb" has a section called current news, where the project highlights the ongoing works. Most of these things are missing from the projects with an increasing entropy trend. More research is needed before drawing any guidelines that could help the OSS projects to keep low entropy as we only looked into the GitHub pages for five projects.

Next, we looked at the correlation between entropy and other project metrics in a time series data. We found that, on average, entropy moderately correlates with total methods, files, and contributors. As contributors are associated with entropy, it will be interesting to investigate how they contribute in increasing entropy, is it their lack of knowledge of software entropy or lack of tool support or lack of knowledge on the codebase, etc. Further research is needed to answer these questions.

Previous research showed that entropy is correlated with bugs [14, 24] and fail to maintain low software entropy will degrade the overall software quality. One of the possible ways to decrease software entropy could be a regular discussion on software design and keep the design document updated at all times. However, for OSS projects, this step will be very hard as developers of OSS projects are geographically distributed, and for it's voluntary characteristics, it is scarce for an OSS project to have updated design documents or in some cases, design documents.

To the best of our knowledge, no popular IDEs report on entropy. As monitoring entropy is essential for projects, developers need a

tool which will monitor entropy of their working source code in real-time and provide developers information about current entropy score and the links between the working class/method to other classes/methods.

5 THREATS TO VALIDITY

Our findings may be subject to the concerns that we list below. We have taken all possible steps to neutralize the impacts of these possible threats, but some could not be mitigated, and it is possible that our mitigation strategies may not have been sufficient.

All the projects in our corpus have been collected from a single source, GitHub. Since we used only GitHub, our findings may be limited to open source projects from GitHub. However, we believe that a large number of projects sampled more than adequately addresses this concern.

All our subject projects are written in Java, but entropy is not specific to Java. Any programs written in a programming language will be prone to a high entropic situation.

Our set of project metrics are only related to source code and developers who write the code. We selected these metrics to see the correlation between entropy and project evolution in terms of its size and number of contributors. However, we can not guarantee that our set of metrics is exhaustive.

We selected the week as our unit of measure for this study, where previous research used different ways of partitioning time (releases, commits, months, year, etc.). However, as our goal was to identify general trends of entropy across projects, week as the unit of measure gives us enough detailed insight into the evolution of projects.

6 RELATED WORK

Entropy is a metric used to measure the naturalness of code. Bianchi et al. [6] have initially introduced software entropy. It represents an overall measure of the degree of quality degradation induced by maintenance interventions, which tend to make code increasingly "chaotic." Tan and Mookerjee [26] analyze the effect of software entropy on maintenance costs on a sample of closed source software projects.

Olague et al. [23] used entropy as one of the metrics to explain the changes that a class undergoes between versions of an object-oriented program. According to the study, high entropic classes tend to change more than classes with lower entropy. Yuet al. [28] combined entropy with component-dependency graphs to measure component cohesion. Entropy was also used by Snider [25] in an empirical study to measure the structural quality of C code.

Chatzigeorgiou et al. [10] proposed entropy is as a design quality metric for object-oriented programs. The difference between the entropy of the two systems provides insight into the quality of the design in terms of how flexible it has been during the enhancement of its functionality.

Entropy has also been used for bug prediction. In a study, Ray et al. [24] found a correlation between buggy code and entropy. According to their study, buggy code has higher entropy than non-buggy code. Hassan et al. [14] compared the entropy with the number of changes and the number of previous bugs in a bug prediction model and found that entropy is a better predictor. D'Ambros et al. [11] extended Hassan's [14] work and found that source code metrics better describe the entropy of changes. Canfora et al. [8] found that the change entropy decreases after any refactoring of the code. Chakraborty et al. [9] used entropy score from statistical language models and used it for spectrum-based bug localization (*SBBL*). They also found a significant improvement compared to standard *SBBL*. In a recent study by Zhang [29], they used cross-entropy with traditional metric suites in a defect prediction model and found that the performance of prediction models is improved by an average of 2.8% in F1-score.

Previous studies have investigated entropy from different perspectives. However, to the best of our knowledge, no studies have investigated the evolution of entropy in OSS projects, and it's relationship with project evolution.

7 CONCLUSION

In this study, we aimed to understand how entropy evolves in OSS projects over time. We study the history of 158 open source projects and found strong evidence that though different types of entropy trends exist, however on average, entropy increases over time. In our corpus, we identified four different entropy trends.

In our study, we also investigate the correlation between entropy evolution and project evolution (in terms of size and number of contributors). To find the correlation between entropy and other project metrics, we conducted cross-correlation analysis as all the data in our corpus are time series. From this analysis, we found that entropy has a moderate correlation with all the project metrics, methods, contributors, and files (see table 4). In our manual inspection of the projects that show a decreasing trend in entropy evolution, we found that the GitHub page of these projects is well documented for not only the user but also for the contributors.

Our work showcases that further research is needed to understand which activities or practices by the project community could help to maintain low entropy over time. Also, the researcher community should try to come up with tools to help the projects to manage the chaos that high entropic code makes over time.

REFERENCES

[1] [n.d.]. Entropy library. https://pypi.python.org/pypi/entropy/0.9//. Accessed: 2017-03-1.

[2] [n.d.]. Shannon entropy. https://en.wiktionary.org/wiki/Shannon_entropy. Accessed: 2017-03-1.

[3] [n.d.]. Understand: Static code analysis tool. https://scitools.com/feature/. Accessed: 2010-09-30.

[4] 2019. Interpret all statistics and graphs for Cross Correlation. https://support.minitab.com/en-us/minitab/18/help-and-how-to/modeling-statistics/time-series/how-to/cross-correlation/interpret-the-results/all-statistics-and-graphs/. Accessed: 2019-08-13.

[5] Iftekhar Ahmed, Umme Ayda Mannan, Rahul Gopinath, and Carlos Jensen. 2015. An empirical study of design degradation: How software projects get worse over time. In *Empirical Software Engineering and Measurement (ESEM), 2015 ACM/IEEE International Symposium on*. IEEE, 1–10.

[6] Alessandro Bianchi, Danilo Caivano, Filippo Lanubile, and Giuseppe Visaggio. 2001. Evaluating software degradation through entropy. In *Proceedings Seventh International Software Metrics Symposium*. IEEE, 210–219.

[7] Grady Booch. 2006. *Object oriented analysis & design with application*. Pearson Education India.

[8] Gerardo Canfora, Luigi Cerulo, Marta Cimitile, and Massimiliano Di Penta. 2014. How changes affect software entropy: an empirical study. *Empirical Software Engineering* 19, 1 (2014), 1–38.

[9] Saikat Chakraborty, Yujian Li, Matt Irvine, Ripon Saha, and Baishakhi Ray. 2018. Entropy Guided Spectrum Based Bug Localization Using Statistical Language Model. *arXiv preprint arXiv:1802.06947* (2018).

[10] Alexander Chatzigeorgiou and George Stephanides. 2003. Entropy as a measure of object-oriented design quality. In *Proceedings of the Balkan Conference in Informatics (BCI)*. 565–573.

[11] Marco D'Ambros, Michele Lanza, and Romain Robbes. 2012. Evaluating defect prediction approaches: a benchmark and an extensive comparison. *Empirical Software Engineering* 17, 4-5 (2012), 531–577.

[12] Mark Gabel and Zhendong Su. 2010. A study of the uniqueness of source code. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*. ACM, 147–156.

[13] GitHub Inc. [n.d.]. Software Repository. http://www.github.com.

[14] Ahmed E Hassan. 2009. Predicting faults using the complexity of code changes. In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*. IEEE, 78–88.

[15] Vincent J Hellendoorn, Premkumar T Devanbu, and Alberto Bacchelli. 2015. Will they like this? evaluating code contributions with language models. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 157–167.

[16] Abram Hindle, Earl T Barr, Zhendong Su, Mark Gabel, and Premkumar Devanbu. 2012. On the naturalness of software. In *Software Engineering (ICSE), 2012 34th International Conference on*. IEEE, 837–847.

[17] C. Izurieta and J. M. Bieman. 2007. How Software Designs Decay: A Pilot Study of Pattern Evolution. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*. 449–451.

[18] C. Izurieta and J. M. Bieman. 2008. Testing Consequences of Grime Buildup in Object Oriented Design Patterns. In *2008 1st International Conference on Software Testing, Verification, and Validation*. 171–179.

[19] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. 2014. The promises and perils of mining GitHub. In *Proceedings of the 11th working conference on mining software repositories*. ACM, 92–101.

[20] Umme Ayda Mannan. [n.d.]. Companion website. http://web.engr.oregonstate.edu/~mannanu/entropyeval/index.html. Accessed: 2020-07-28.

[21] S. McIntosh. 2011. Build system maintenance. In *2011 33rd International Conference on Software Engineering (ICSE)*. 1167–1169.

[22] Andrew V Metcalfe and Paul SP Cowpertwait. 2009. *Introductory time series with R*. Springer.

[23] Hector M Olague, Letha H Etzkorn, and Glenn W Cox. 2006. An Entropy-Based Approach to Assessing Object-Oriented Software Maintainability and Degradation-A Method and Case Study.. In *Software Engineering Research and Practice*. 442–452.

[24] Baishakhi Ray, Vincent Hellendoorn, Saheel Godhane, Zhaopeng Tu, Alberto Bacchelli, and Premkumar Devanbu. 2016. On the naturalness of buggy code. In *Proceedings of the 38th International Conference on Software Engineering*. ACM, 428–439.

[25] Greg Snider. 2001. Measuring the entropy of large software systems. *HP Laboratories Palo Alto, Tech. Rep* (2001).

[26] Yong Tan and Vijay S Mookerjee. 2005. Comparing uniform and flexible policies for software maintenance and replacement. *IEEE Transactions on Software Engineering* 31, 3 (2005), 238–255.

[27] TIOBE. [n.d.]. TIOBE Index. https://www.tiobe.com/tiobe-index/.

[28] Yong Yu, Tong Li, Na Zhao, and Fei Dai. 2008. An approach to measuring the component cohesion based on structure entropy. In *2008 Second International Symposium on Intelligent Information Technology Application*, Vol. 3. IEEE, 697–700.

[29] Xian Zhang, Kerong Ben, and Jie Zeng. 2018. Cross-entropy: A new metric for software defect prediction. In *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 111–122.