Repo Velocity: Diagnosing Git Repository Health

Tyler Brown
NUID: 001684955

1 Introduction

Understanding the health of open source projects is important to industry because it helps them assess risk associated with their technology stack. Forecasting health is important to investors because this information can help them make profitable investments in open source. Academia is interested to know if there are links between theory, such as programming language design [1], and the health of open source software. A previous metric used to assess open source software health is the Truck Factor [2]. Truck Factor is the smallest subset size of developers who contributed 50% of the code in an open source project. The underlying intuition is that open source projects with a lower Truck Factor are more susceptible to project disruption in the event of adverse circumstances.

By borrowing from Physics, this paper [1] contributes to the advancement of understanding the health of open source projects, studying a subset of GitHub repositories, with the following:

- Introducing a health measure that can be assessed at time $t_{i\pm k}$ where k is a multiple of i.
- New health measure can be used in forecasting as well as description
- Health measure is rooted in Physics so we can derive related measures using preexisting theory.

2 Related Work

Truck Factor reflects robustness of project [2] by computing the minimum number of developers required to comprise 50% of file ownership. The underlying intuition is that a project with a low number of dominant developers will be more susceptible to failing due to external shocks. Projects where file ownership is shared by a large number of developers are interpreted as more healthy with this approach.

Not all projects reflects software which requires support. Researchers have classified many project types such as 'software development projects', 'solutions for homework', 'projects with educational purposes', 'data sets', and 'personal web sites' [3]. The observation that many projects are not software development is also a noted peril when analyzing GitHub [4]. We may also find repositories containing code duplicates [5]. These factors may comprise threats to validity which are important to address in this analysis.

3 Methods

The Truck Factor assesses Git repository health at time t_i. We would like to better understand health at different points in time $t_{i\pm k}$ where $k \geq 1$. We can borrow from Physics to find such a measure. Velocity, $v(t)$, is a measure of distance over time. What is a proxy for distance in the context of Git

[1] Code available on GitHub: https://github.com/tbonza/EDS19

repository health? Total lines of code changed in a project during time t_i can tell us how quickly the code base is changing. We apply this definition of velocity, $v(t)$, for our new measure Repo Velocity.

$$\text{Repo Velocity } \approx v(t) = \frac{d}{t} = \frac{\text{lines added } - \text{ lines deleted}}{t} \qquad \forall i \pm k \in t | i \in \mathbb{R}, k \in \mathbb{R}_{>0} \quad (1)$$

Equation 1 defines our target measure, Repo Velocity, and posits that the Physics concept of 'distance' can provide insight into how a GitHub repository is changing over time. We can compute velocity, $v(t_i)$ at time t_i where i is a real number. We can also compute average velocity, $v_{avg}(t_{i \pm k})$, at time t_i to time $t_{i \pm k}$. The Repo Velocity measure also has the advantage of allowing aggregation and disaggregation. Truck Factor is a measure which can only be analyzed at an aggregate level. There can be a lot of interesting variation below a Repository level aggregation. Repo Velocity can also be used to capture individual contributor velocity. The ability of Repo Velocity to be applied over time as well as at aggregated and disaggregated levels make the measure a unique contribution to the understanding of Git repository health. The remaining methodology seeks to establish Repo Velocity as an informative descriptive and predictive target or dependent variable.

3.1 Establishing Baseline Repo Velocity

In our effort to establish a new measure which can be applied at time

3.1.1 Descriptive Baseline

got some trends

3.1.2 Predictive Baseline

using a simple prediction

3.2 Validating Repo Velocity at Scale

Collecting from Facebook, etc.

3.3 Data Engineering

Wrote a Python package, Okra.

3.3.1 Collecting Data

used kubernetes and “being nice”

3.3.2 Processing Data

using Apache spark, using “hack/reduce”

4 Experiment

4.1 Results from Baseline Repo Velocity

4.1.1 Results from Predictive Baseline

4.1.2 Results from Scaled up Prediction

4.2 Results from Repo Velocity at Scale

4.2.1 Modeling Outliers

bad

5 Discussion

had mixed results, should have better classified input GIGO

5.1 Threats to Validity

yup

References

[1] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. A large scale study of programming languages and code quality in github. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 155–165. ACM, 2014.

[2] Guilherme Avelino, Marco Tulio Valente, and Andre Hora. What is the truck factor of popular github applications? a first assessment. Technical report, PeerJ PrePrints, 2015.

[3] Marcus Soll and Malte Vosgerau. Classifyhub: An algorithm to classify github repositories. In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, pages 373–379. Springer, 2017.

[4] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. An in-depth study of the promises and perils of mining github. *Empirical Software Engineering*, 21(5):2035–2071, 2016.

[5] Cristina V Lopes, Petr Maj, Pedro Martins, Vaibhav Saini, Di Yang, Jakub Zitny, Hitesh Sajnani, and Jan Vitek. Déjàvu: a map of code duplicates on github. *Proceedings of the ACM on Programming Languages*, 1(OOPSLA):84, 2017.