2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)

A Time Series Analysis of TravisTorrent Builds: To Everything There is a Season

Abigail Atchison, Christina Berardi, Natalie Best, Elizabeth Stevens, Erik Linstead†
Machine Learning and Assistive Technology Lab, Schmid College of Science and Technology
Chapman University
Orange, CA, USA
†linstead@chapman.edu

***Abstract*—We apply a seasonal decomposition time series analysis to TravisTorrent data in order to examine growth trends and periodic behavior related to number of builds in a continuous integration environment. We apply our techniques at the macro level using the full TravisTorrent repository consisting of 1,283 projects, and at the micro level considering the Apache Drill project. Our results demonstrate strong seasonal behavior at both the large and small scale using an additive time series model. In addition to being able to accurately capture trend and periodicity in build data, our techniques are also able to accurately forecast the expected number of builds for a future time interval.**

***Keywords*-TravisTorrent; Time Series; Build Data; Data Seasonality**

I. Introduction

In recent years, the availability of scalable, web-based tools has facilitated the adoption of continuous integration (CI) as part of the software development lifecycle for open source and proprietary projects alike. In theory, CI brings a host of benefits to the development process, including early detection of bugs, automated testing, simplified releases, and the incentive to commit code to versioning systems early and often [1]. In practice, CI can be, and often is, tailored to fit individual organizational needs [2], and has been demonstrated to lead to improved code quality and developer productivity [3].

While previous studies have focused on the cost and benefits of CI as a tool to support incremental software development paradigms, such as agile, the availability of a large repository of historical build data allows us to examine the relationship in the other direction. In particular, how do the day-to-day routines of software developers and the battle rhythms of software projects manifest themselves in build activity in a CI system? Does this build activity exhibit periodicity over time, and can this periodicity be detected with machine learning? Similarly, how does the overall growth and maturation of a software project impact the use of CI builds? In this paper we apply time series analysis to the TravisTorrent data set in order to gain insight into these questions, with the ultimate goal of understanding the underlying temporal patterns of software development. We carry out a seasonal decomposition of build history on both the large and small scale, finding distinct cyclic behavior. Finally, we apply time series forecasting to explore the feasibility of such algorithms to predict build activity in the future, with a promising initial result.

II. Data

We base our time series analysis on the data provided in the February 8, 2017 release of TravisTorrent [4]. This repository consists of 3,702,595 analyzed builds from 1,283 open source software projects over a 5 year timespan. On average, each project contains approximately 2,886 builds, with a minimum of 2 builds for the Puppet OSX and a maximum of 471,417 builds for the Rails project.

In addition to analyzing temporal trends on the large scale using the entire TravisTorrent dataset, we also consider the small scale, consisting of a single project. For this analysis we chose the Apache Drill project [5], a popular framework in the big data community, used to issue highly-distributed queries to analyze NoSQL databases. Drill consists of 845 builds over a timespan of almost 2.5 years, and thus provides a sufficiently rich history to be a good candidate for time series analysis. It serves as a proof of concept for modeling at the level of an individual project, though of course our technique is automated and can be easily applied to all projects in the dataset.

III. Methods

Given the raw TravisTorrent data, we start by aggregating the number of builds on a daily basis using the gh_build_started attribute in the TravisTorrent database schema. For the full data set this results in 1,830 non-zero data points for the time period starting August 29, 2011 and ending August 31, 2016. Similarly, for the Apache Drill project, this results in 442 non-zero data points for the time period starting March 4, 2014 and ending August 30, 2016.

For each of these data sets, we order the data points temporally to create a time series representation, D_t. In order to explore the underlying structure of this data, we assume it is described by the following additive model: $D_t = T_t + S_t + R_t$. This assumption allows us to treat D_t as the sum of a trend component, T_t, a seasonal component, S_t, and a random component, R_t. By examining each of these components, especially T_t and S_t, we are able to isolate global temporal patterns related to long term growth and decline over the entire timespan, as well as local periodic patterns being driven by repetition over time.

While several techniques exist for decomposing a time series into its respective components, we rely on a popular

978-1-5386-1544-7/17 $31.00 © 2017 IEEE
DOI 10.1109/MSR.2017.29

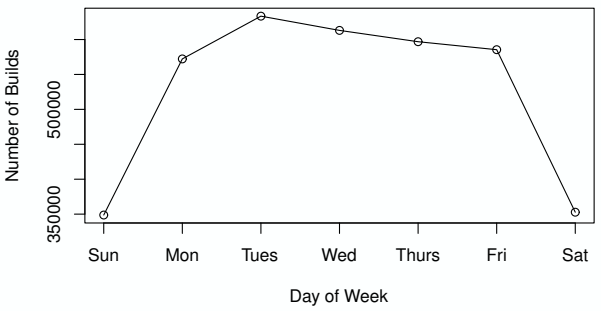

Fig. 1. Total Builds by Day of Week

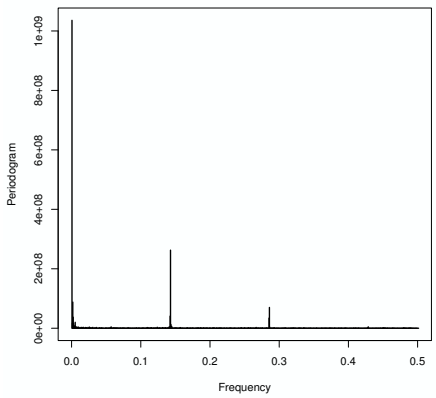

Fig. 2. Periodigram of TravisTorrent Builds

algorithm, STL [6], for this study. STL, or seasonal and trend decomposition using loess, employs a sequence of smoothing operations using loess, or local regression. Loess is unique in that it does not assume the data must fit any specific shape. This is achieved by locally fitting a curve at a point by using a subset of neighboring data points. Overall, STL is made up of a set of nested loops. The inner loop updates the seasonal and trend components with each pass. The outer loop computes the weights for neighboring data points, which are used in the next iteration of the inner loop to calculate the seasonal and trend curves. An efficient implementation of STL is readily available in the R statistics package [7].

To predict future build activity based on a given time series, we employ an AutoRegressive Integrated Moving Average (ARIMA) model [8], a standard forecasting algorithm in domains such as finance and econometrics. Given sufficient data, ARIMA is capable of handling both seasonal and non-seasonal trends, and can be tuned to adapt to non-stationary time series. An ARIMA model, as its name implies, consists of three components. The autoregressive component predicts future values of an observed variable based on past observations, in combination with the moving average component, which smooths error. Finally, the integration component handles non-stationarity, as needed. Together these components can model complex time series with high fidelity compared to less sophisticated techniques. A popular version of ARIMA is also available in the R forecast package [9].

IV. Results

We start our analysis at the macro level by considering the full TravisTorrent data set. Our hypothesis is that the number of builds on a daily basis is explained by an additive time series. More specifically, we hypothesize that in addition to a general trend driven by individual project milestones (such as formal releases), the number of build jobs exhibits substantial periodicity. Informally, this can be investigated via exploratory data analysis. Figure 1 shows, for the entire 5 year history of the dataset, the number of build jobs aggregated by days of the week. The figure indicates clear spikes at the beginning of the work week, followed by a subtle decline toward the end of the work week, followed by a drastic decline in build jobs on the weekend (Saturday and Sunday). From this, one can speculate that the number of builds follows weekly seasonality that will manifest itself in an additive time series model.

While exploratory data analysis is useful to get a coarse-grained estimate of seasonality, we leverage more formal methods to confirm mathematically that such periodicity exists. This is achieved by taking the discrete time Fourier transform (DTFT) of the time series to convert from the time domain to the frequency domain. The distribution of these frequencies can then be examined to determine the dominant cycle of the data. Figure 2 depicts the power spectrum of the TravisTorrent time series. A dominant frequency is clearly seen at 0.143 Hz. By dividing 1 by this number we can convert into days, which yields a value of 6.99, almost exactly 1 week! Thus, our intuition that TravisTorrent exhibits weekly seasonality is confirmed by the DTFT, which can be fed to the STL algorithm to accurately decompose our additive time series into its individual components.

After determining periodicity of the TravisTorrent time series, we then decompose the series into its trend, seasonal, and stochastic components. Figure 3 provides a visual representation of this decomposition. In examining the trend line, we clearly see steady growth in number of builds from the start of data collection in 2011 that continues to the end of 2014 and beginning of 2015, at which time there is a brief plateau. This is followed by a subtle but noticeable decrease for the remainder of 2015. This trend continues for the remaining data available in 2016, though during this time period the trend line appears to start leveling out. We hypothesize that this is due to the fact that the majority of the almost 1,300 projects in the TravisTorrent dataset, at the time of writing, are mature, "popular" projects that are converging on a steady state as part of their software development lifecycle.

Having performed our time series analysis on the large scale across 1,283 projects, we turn our attention to the small scale

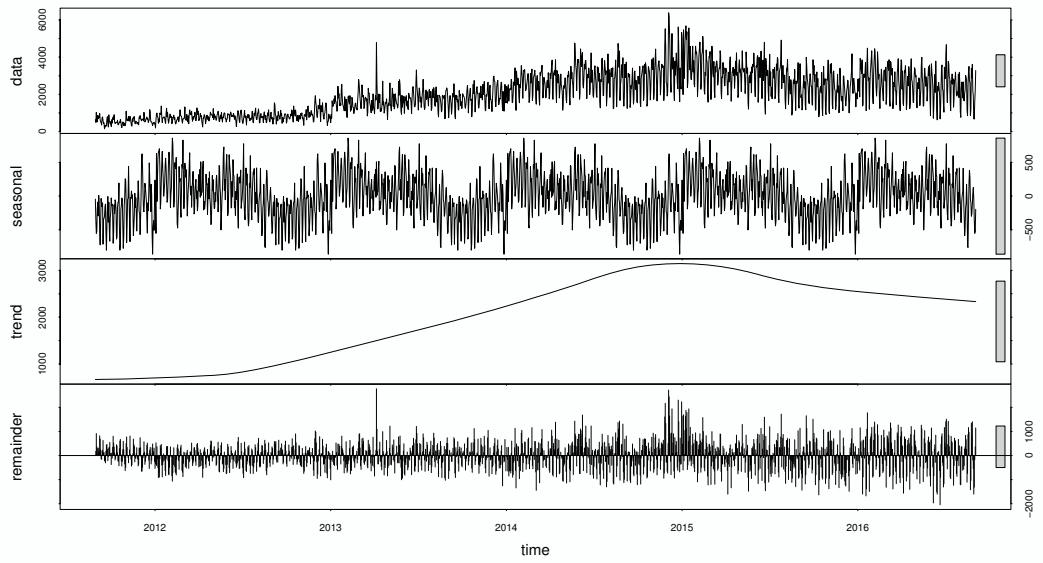

Fig. 3. Time Series Decomposition of all TravisTorrent Builds

in order analyze the build history of Apache Drill. We again start by examining the power spectrum of the time series. Unlike the full data set, the power spectrum of Apache Drill indicates a dominant frequency of about 31 days, or 1 month, and so we aggregate the data on a monthly basis to be able to better visualize this trend.

Figure 4 depicts the decomposition of the Drill time series. Again, the trend component shows a clear period of growth in the number of builds, which reaches its maximum around May 2015. This is significant as the release notes for the project indicate that a major milestone release, version 1.0.0, occurred on May 19, 2015, following several minor releases. Further inspection of the release notes also shows that this version of the software included a disproportionately large number of bug fixes and feature integrations, which no doubt drove the corresponding increase in build jobs. After this major release we see a steady decline in the number of builds, which continues throughout the remainder of the data, despite the fact that Drill had eight minor releases (versions 1.1.0-1.8.0) between June 2015 and August 2016.

In addition to the trend component of the time series, clear periodic behavior is seen at both the large and small scale, as evidenced in figures 3 and 4. For the full TravisTorrent data set we see repeated weekly behavior resulting in peaks and valleys over the course of each year. In addition to the weekly cycle, we are also able to detect a longer-term, and more subtle, trend which appears to correlate with a dip in build activity near the end of the calendar year and picking up again after the start of the calendar year. We hypothesize that this corresponds to the fall holiday season. Moving to the seasonal component of Apache Drill, we again see a clear pattern, though one that is strikingly different from the full data set. This is due in part to the difference in periodicity (monthly for Drill, weekly for all data), and the relative quantities of data available at each level of granularity. Looking over the course of a single year for Drill, the build activity exhibits two peaks near the start of the year, followed by a valley in the middle of the year, climbing into two more peaks at the end of the year.

While the decomposition of build data time series is interesting in its own right to understand seasonal and non-seasonal patterns in historical software development activity, this approach also suggests that we can exploit such structure to predict future build activity. To test this hypothesis we built a seasonal ARIMA model based on the available Apache Drill data, but excluded the last 3 months of data (June, July, and August 2016) as part of the training phase. We then forecasted an estimate for the number of builds expected for these months. The ARIMA model predicted 15, 20, and 20 builds respectively, compared to the 17, 16, and 18 builds actually observed for this time period, as presented in figure 5. This represents an average accuracy of 86%, and while this only represents a single experiment, suggests that build activity and time series forecasting could be used to accurately predict software release milestones and related events using nothing but continuous integration data by leveraging temporal trend and seasonality.

V. Related Work and Future Directions

Our research aims to investigate build data in order to identify patterns in build volume over the course of five years, both at the micro and macro level. Although we chose the generalized approach of investigating all builds over time, others have delved into the realm of using specific aspects of build data as a tool to identify temporal patterns. Sliwerski et

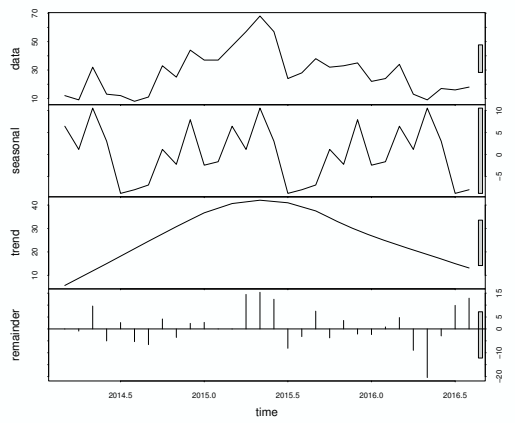

Fig. 4. Time Series Decomposition of Apache Drill Builds

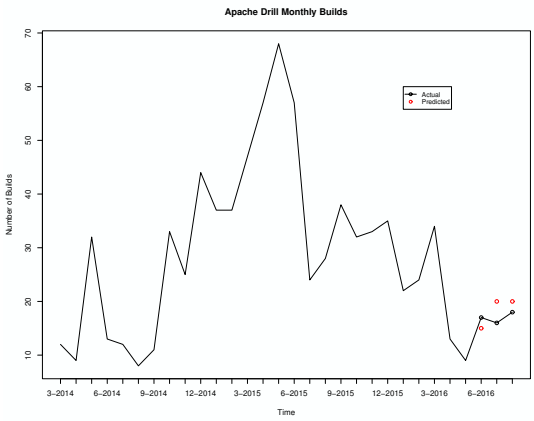

Fig. 5. Forecasting Apache Drill Monthly Builds

al. linked bug databases to their corresponding version archive in order to gain deeper insight into which days of the week more commonly resulted in changes that would later need to be fixed. They concluded that the likelihood that a change to the code databases they studied would later need to be fixed (and was thus a bug) was highest on Fridays [10]. However Eyolfson et al. concluded that day of the week bugginess is only consistent on a per project basis, and thus cannot be used to create universal bugginess predictions. They instead delved into other aspects that could influence bugginess, including time of day, contributor seniority, and the frequency with which the contributor committed code [11].

Throughout this exploration, we utilized time series analysis to identify temporal patterns within a dataset. Raja et al. constructed a computationally efficient and easy to apply model that conducted the first time series analysis across multiple projects and organizations in order to look at software progression through the lens of defects across code versions [12]. Their model has been used to facilitate planning and time allocation when looking at adding changes to a code base. Fuentetaja et al. applied time series analysis to the study of historical software systems and their evolution in terms of software metrics [13]. They ultimately conclude that software is feedback driven, a conclusion that adds further weight to the notion of seasonality in build activity.

One future research direction that could provide additional insight into the build data we have analyzed would be to aggregate data from and execute a time series analysis on user forum posts, such as those found on Stack Overflow. This potentially new dimension of temporal patterns would allow us to further investigate the idea that software is feedback driven through the process of collaborative development practices. Despite our identification of a clear seasonality in build activity, we would be interested in understanding whether this seasonality is mirrored in the discussions that inevitably contribute to bug fixes or feature additions that in turn feed the continuous integration process.

REFERENCES

[1] M. Fowler and M. Foemmel, "Continuous integration, http://www.martinfowler.com/articles/continuousIntegration.html," 2005.

[2] D. Sthl and J. Bosch, "Modeling continuous integration practice differences in industry software development," *Journal of Systems and Software*, vol. 87, pp. 48 – 59, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121213002276

[3] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, "Quality and productivity outcomes relating to continuous integration in github," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: ACM, 2015, pp. 805–816. [Online]. Available: http://doi.acm.org/10.1145/2786805.2786850

[4] M. Beller, G. Gousios, and A. Zaidman, "Travistorrent: Synthesizing travis ci and github for full-stack research on continuous integration," in *Proceedings of the 14th working conference on mining software repositories*, 2017.

[5] "Apache drill." [Online]. Available: https://drill.apache.org/

[6] R. B. Cleveland, W. S. Cleveland, J. E. Mcrae, and I. Terpenning, "STL: A Seasonal-Trend Decomposition Procedure Based on Loess," *Journal of Official Statistics*, vol. 6, no. 1, pp. 3–73, 1990. [Online]. Available: http://www.jos.nu/Articles/abstract.asp?article=613

[7] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2016. [Online]. Available: https://www.R-project.org/

[8] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[9] R. Hyndman and Y. Khandakar, "Automatic time series forecasting: The forecast package for r," *Journal of Statistical Software*, vol. 27, no. 1, pp. 1–22, 2008. [Online]. Available: https://www.jstatsoft.org/index.php/jss/article/view/v027i03

[10] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" *SIGSOFT Softw. Eng. Notes*, vol. 30, no. 4, pp. 1–5, May 2005. [Online]. Available: http://doi.acm.org/10.1145/1082983.1083147

[11] J. Eyolfson, L. Tan, and P. Lam, "Correlations between bugginess and time-based commit characteristics," *Empirical Software Engineering*, vol. 19, no. 4, pp. 1009–1039, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10664-013-9245-0

[12] U. Raja, D. P. Hale, and J. E. Hale, "Modeling software evolution defects: a time series approach," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 21, no. 1, pp. 49–71, 2009. [Online]. Available: http://dx.doi.org/10.1002/smr.398

[13] E. Fuentetaja and D. J. Bagert, "Software evolution from a time-series perspective," in *International Conference on Software Maintenance, 2002. Proceedings.*, 2002, pp. 226–229.