Mining Recurrent Activities: Fourier Analysis of Change Events

Abram Hindle
University of Waterloo
Waterloo, Ontario
Canada
ahindle@cs.uwaterloo.ca

Michael W. Godfrey
University of Waterloo
Waterloo, Ontario
Canada
migod@cs.uwaterloo.ca

Richard C. Holt
University of Waterloo
Waterloo, Ontario
Canada
holt@cs.uwaterloo.ca

Abstract

Within the field of software repository mining, it is common practice to extract change-events from source control systems and then abstract these events to allow for different analyses. One approach is to apply time-series analysis by aggregating these events into signals. Time-series analysis requires that researchers specify a period of study; usually "natural" periods such as days, months, and years are chosen. As yet there has been no research to validate that these assumptions are reasonable. We address this by applying Fourier analysis to discover the "natural" periodicities of software development. Fourier analysis can detect and determine the periodicity of repeating events. Fourier transforms represent signals as linear combinations of sine-waves that suggest how much activity occurs at certain frequencies. If behaviors of different frequencies are mixed into one signal, they can be separated. Thus Fourier transforms can help us identify significant development process sub-signals within software projects.

Figure 1. Time Domain, Frequency Domain, and Spectrogram of a whistle and a word.

1 Introduction

Significant aspects of human behavior can be characterized by repeating patterns and processes, such as waking, eating, sleeping and even working. We noticed repeating patterns of development, such as slow weekends or declining work quality on Fridays [4], in many software repositories.

The data stored in a source control system (SCS), such as CVS or Subversion, is complex; it consists of thousands of events recorded by many actors potentially working in parallel, often in different time-zones. When we abstract these events into time-series we observe much noise in the time-series. It is not clear whether breaking down the activity into natural time units such as days or weeks reveals all the recurrent behavior of interest. Time-series analysis relies on the assumption of a periodicity that is supplied by the user l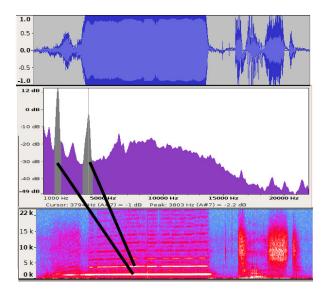ooking at the data. Analysts will choose or guess a period length and then analyze the time series with respect to that period. Fourier analysis does not try to assume such a periodicity; instead, it reflects the periodicity of the behavior by the magnitude of activity taking place at a certain frequency.

Thus we propose the use of the Fourier transform to allow us to analyze data without assuming or guessing a periodicity that exists in the data. Fourier transforms can help us by detecting the repeating and cyclic behaviors inside the signal.

Our hypothesis is that interesting repeating signals exist within the SCS and CVS repositories, we can detect and potentially identify these signals using the Fourier transform.

1.1 Motivation

Before we started this investigation we assumed that we could detect and see some frequent behaviors in a SCS with

a Fourier transform. We anticipated that we could detect signals that were more complicated and more interesting than signals of daily commits. We expected that the Fourier transform would help us identify intentional, process-driven recurrent activities and natural, time driven, recurrent activities. For instance if one day of a week was used for meetings, we would expect there would be a noticeable pattern with a frequency of one week.

We hope some repositories will exhibit recurrent behaviors that are composed of events at different frequencies, such as the frequency of code review meetings versus general development, or perhaps the end of an iteration. Fourier transforms are especially good at separating out spectral components of a signal, such as behaviors occurring concurrently at different frequencies.

In our studies we will use the Discrete Fourier Transform (DFT). In this paper we will be applying the DFT to counts of revisions per time period.

Figure 1 depicts an audio signal depicted of a whistle followed by the word "test". The whistle has harmonic components (the streaking lines seen in the spectrogram); a human speaking the word "test" will consist of noise from the consonants. In Figure 1 we can see the frequency domain plot in the middle. Two peaks are highlighted, one main peak and a secondary peak. These peaks indicate two strong frequencies in the signal, specifically the frequencies of the whistling. The two strong frequencies are not noticeable in the time/amplitude domain plot shown on top, but these signals are noticeable in the spectrogram just below it. The two bottom light colored streaks across the first half of the signal show the dominant frequencies of the whistle. The largest streaks correlate with the two peaks in the frequency domain. We suspect that this kind of observation and correlation should work on signals extracted from SCSs.

Figure 2 illustrates how a Fourier transform can be used to analyze a signal. We have two real signals, arising from a tester and a main developer. The main developer is making commits every two days, the tester is committing their test code every four. It would be difficult to detect these two mixed sub-signals from the time-series alone. Using the Fourier transform we notice that the time-series can be created from the linear combination of a sine waves with periods of two and four days. This shows that we can observe the two signals that exist within the repository that were difficult to separate in the time domain but easy to spot and separate in the frequency domain.

The purpose of this paper is an exploratory investigation to determine how to apply the Fourier transforms to some of our Software Engineering related problems.

1.2 Previous Work

Within the Software repository mining community, there has been work by Herraiz et al. [2] on using Time Series Analysis, ARIMA models, and autocorrelation on SCS data. Herraiz studied many OSS projects and can characterize the ARIMA model's range parameters used to model these projects.

LPC was applied by Antoniol et al. [1], which is similar to the Fourier transform but was not used to find time-invariant relationships.

We have done previous work with time series and characterizing the behavior in a repository by splitting the kinds of revisions by their likely task [3].

2 Fourier Transform

In our work, we adapt Fourier analysis to mining logged information about software development. For our purposes a Fourier transform is a black box. We give it a time-series, a signal, as input and it outputs the dominant frequencies of that signal. We can use this output to help us identify the behaviors that produced these signals. We can use the output of a Fourier transform to see if there exists any high, low or medium frequency behaviors common across the entire development of the project, or even smaller periods such as a month. Alternatively, multiple smaller Fourier transforms of shorter time intervals can be combined into a spectrogram, which shows the changing behavior with respect to frequency over time (see Figure 4).

The Fourier transform takes a time-domain function and converts it into a frequency-domain function. This means we can take a signal and convert it from a time / amplitude signal to a frequency / magnitude signal. A Fourier transform, of size n, outputs a series of coefficients for sine-waves with frequencies of 0 to n. The magnitude of these coefficients indicates how strongly a sine-wave of that frequency exists in that signal. Fourier transforms are often used in frequency analysis to analyze and plot the harmonic or frequency based components of signals.

The signals that we analyze are often events per unit of time, such as revisions per day over a range of days. With an input of a window of revisions per day, the Fourier transform outputs a frequency versus magnitude representation of that window, as coefficients of sine waves with different frequencies (from low frequency to high frequency). The summation of these weighted sine waves produces the signal of the window in the time-domain. Thus the frequency representation of a signal does not lose any information, in fact it represents the same signal. We can use this to our advantage, since we can get the inverse of a Fourier transform so we can filter and produce signals as well.

We use the Discrete Fourier Transform (DFT), this is the most common Fourier transform used on discrete data. The input to a discrete Fourier transform is a sequence of n real or complex samples of a signal.

Sometimes a Fourier transform can be made less noisy by multiplying the signal by a windowing function. Win-

Frequent   Infrequent   Sum of Frequent and Infrequent

Time-Domain of Revisions per day

Sinusodial Composition

Fourier Transform: Frequency Domain of Revisions Per Day

Day 1   Day 3   Day 5   Day 7
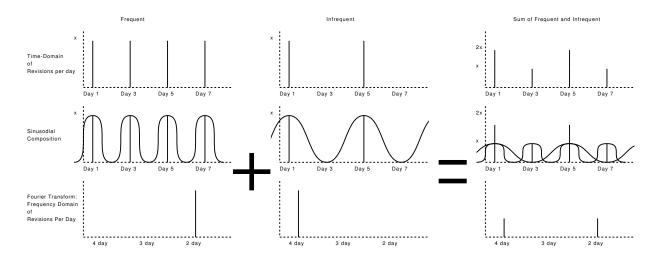
4 day   3 day   2 day

Figure 2. Example of how a Fourier transform can be used to analyze a signal

dowing functions commonly used consist include the Hamming window, and the Hanning window, which look similar to the first half of a sine wave. Window functions are used to help analyze data by massaging the data into a form that seems more continuous and removes the discontinuity at the beginning and the end of the signal, which reduces noise.

The Fourier transform is additive and this translates between both frequency and time domains. Two signals, with frequencies of k and l, when added together in the time domain will produce a Fourier transform where frequencies k and l have high magnitude. The additive property means we can compose and take apart signals, thus we can analyze subparts of a signal before hand, and then combine them to observe the aggregated behavior.

2.1 Discrete Fourier Transform

Figure 3 depicts a Discrete Fourier Transform (DFT) executed across the entire lifetime of Max DB 7.500, an Open Source database system maintained by MySQL. Max DB 7.500 was the first major version of Max DB that was released as Open Source. Large peaks stand out in Figure 3, they indicate a periodic, globally distinguishable, behavior within the Max DB 7.500 repository. We can see that the plot does not follow a power-law-like curve, which is a common Fourier transform pattern that often indicates either much low frequency activity or much noise. Thus this plot is promising because it indicates potential frequent and periodic behavior, it is not all skewed to low frequency noise.

Before one looks for behaviors with a Fourier transform, one must decide what is the maximum frequency, or conversely the maximum length, of the behavior sought. A Fourier transform operating on N time units can only clearly represent frequencies between 0 and $N/2$. Any frequencies over $N/2$ will be aliased by a frequency of $x-N/2$

where x was the frequency above $N/2$. $N/2$ is called the Nyquist frequency, it is the maximum frequency we can measure with a Fourier Transform of N bins. Although Fourier transforms can operate on any number of bins, it is often faster to choose bins that are powers of two.

If the number of bins is not a power of two we can pad both the start and end of the data with zeroes until the length of the signal, the data matches a power of two. If the analyzed signal fits into bins that are a power of two, then we can apply the Fast Fourier Transform (FFT), which runs in $O(NlogN)$ instead of $O(N^2)$.

Since our time units will be discrete, the counts and values we measure will exist over the real numbers. Fourier transforms operate on complex numbers so we shall assume that all our inputs have an imaginary component of $0i$.

Once the signal is passed through the Fourier Transform, it is transformed into $n + 1$ coefficients. Coefficients 0 to $n/2$ will contain the coefficients for sine-waves of frequency $x/(n * t)$ where x is a bin from 0 to n, and t is length in time of the signal. Those bins with larger magnitudes (where a bin x value is $a + bi$, $magnitude(a + bi) = \sqrt{a^2 + b^2}$) indicate that sine-waves of those frequencies are prominent in the signal. Depending on the shape of the recurrent behavior, it could smear across multiple bins, but if the shape of the signal was truly sinusoidal it should appear prominently in one of the bins.

2.2 Spectrograms

Figure 4 illustrates a spectrogram where the magnitude of the frequency is represented by color, time is along the horizontal axis and frequency along the vertical. Spectrograms are formed by taking windows of events and applying the Fourier transform to each window. The Fourier transforms of these windows are plotted together in time-ordered sequence, the first axis is time window, the second axis is
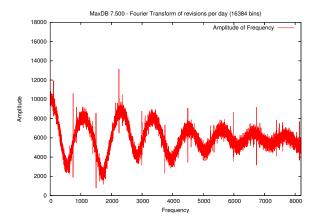
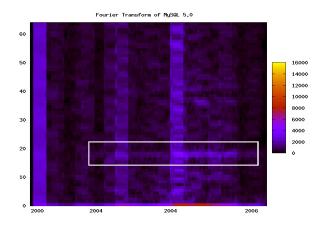**Figure 3. Fourier Transforms of MaxDB 7.500**

Figure 4. Spectrogram of MySQL 5.0

frequency and the third access is amplitude. The windows are centered on the time in question, often causing the windows to overlap. Figure 4, which is of MySQL 5.0, has a horizontal smear at a frequency of 18 or 19, we suspect this behavior represents the cycle of a development week. We found that this behavior included interactions from accounts that were not in the top-ten most frequent committing accounts.

2.3 Applied to Revision Events

In this paper, our goal is to apply frequency analysis, using the Fourier transform, to software engineering data such as the daily records of maintenance activities. We expect that recurring patterns of behaviors exist within many projects.

Behaviors we anticipate would include large or small check-ins done on Fridays, or cases where no work is done on Saturdays. We could recognize if programmers only work from 9 to 5. These behaviors should not be difficult to spot, perhaps other recurrent behaviors exist. We hope to

detect recurrent behaviors such as weekly or bi-weekly code audits, or large change events that occur frequently within an SCS.

We did preliminary work and looked for repeating behaviors in revision data and we found many examples. We often observed large, noisy spikes consisting of many frequencies, usually each spike is correlated with a period that had many changes. We wanted to look for smears of frequencies across time, such as those produced by the whistling in Figure 1, or the smear in Figure 4, as these smears would indicate a repeating process.

We extracted change data from many projects. We have observed horizontal smears across Fourier transforms for the following projects: Mozilla, MySQL, Evolution, MaxDB, and Xerces. These smears indicate a recurring activity during the development of these projects.

3   Conclusions

We have shown that the Fourier transform can be used to uncover repeating patterns or processes that are hidden in the logs of change events. Using the Fourier transform we can analyze the behavior of these signals to isolate their frequencies and their recurrent behaviors.

We have observed that some Open Source project behavior is externally variable but still internally consistent. We observed repeating behaviors in a sample of large Open Source systems such as MySQL and MaxDB.

Current and future work includes working with auto-correlation and self-similarity of Fourier transforms to partition time by recurrent development behavior. We are also leveraging the use of the Fourier Transform on other software engineering related data-sources such as traces and server logs.

References

[1] G. Antoniol, V. F. Rollo, and G. Venturi. Linear predictive coding and cepstrum coefficients for mining time variant information from software repositories. In *MSR '05: Proceedings of the 2005 international workshop on Mining software repositories*, pages 1–5, New York, NY, USA, 2005. ACM.

[2] I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles. Forecasting the number of changes in eclipse using time series analysis. In *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 32, Washington, DC, USA, 2007. IEEE Computer Society.

[3] A. Hindle, M. W. Godfrey, and R. C. Holt. Release pattern discovery via partitioning: Methodology and case study. In *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 19, Washington, DC, USA, 2007. IEEE Computer Society.

[4] J. Śliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? In *MSR '05: Proceedings of the 2005 international workshop on Mining software repositories*, pages 1–5, New York, NY, USA, 2005. ACM.