Using peak analysis for identifying lagged effects between software metrics

Josée Tassé
Dept. of Computer Science and Applied Statistics
University of New Brunswick, Saint John campus
Saint John, New Brunswick, Canada
jtasse@unbsj.ca

***Abstract*— Measures extracted from software repositories tend to be collected on a regular basis (often daily), forming time series of data. In this context, it is normal to assume that some of the measures collected are having an effect on other measures, potentially with some delay (or "lag") in the effect. Such delay in the effect may even vary over time, making the identification of the effect difficult. In this paper, we present our initial ideas on a simple analysis method for such situation, in which peaks from the two time series in question are analyzed, and similar ones are matched.**

Keywords-Peak analysis; time series; lagged effect; software metrics

I. Introduction

Measures extracted from software repositories tend to be collected on a regular basis. For example, one could capture the number of bugs found and the number of bugs fixed every day or every week. In statistical terms, such sequence of data collected regularly over time form a time series.

Data are usually kept in this format (time series) only if they are simply displayed graphically (showing their evolution over time) or used in statistical control. For other analyses, data are typically transformed, losing most of their time-related information. In particular, one might aggregate such measures into a single one per version or per time segment (e.g., total number of bugs found, or total number of lines of code changed, during the development of one version). Then, only the differences between two consecutive versions or time segments are used in the analysis. For example, the expected number of bugs to be found during the development of a particular version may be predicted based on the number of lines of code changed in the previous version (and/or other variables of interest – those are surveyed in [1]). The problem here is that by limiting the association to only two consecutive time segments, it is not possible to detect a longer effect (e.g., what if a change in the number of lines of code changed was having an effect up to three releases later?).

What is needed is a way to analyze the metrics in question as time series, showing if one metric is causing another one to change, and if yes, after how much time can such impact be felt. Let's illustrate this with the problem of judging the stability of a system or sub-system. What is really needed here is to characterize what happened in the past (e.g. amount of change) that eventually lead to a decrease in the stability (perhaps measured as bugginess). Being able to analyze the lagged effect of the amount of change, or other metrics of interest, on the bugginess (or other indicators of stability), could help identify appropriate criteria for decisions related to software stability. If the lag is important enough to allow for a reaction time, this could lead to early warnings about upcoming problems with the stability unless something is done to reduce – or even eliminate – the problem. Depending on the type of changes done (e.g., corrective vs. perfective maintenance) or the complexity of the changes being made, it may take more or less time to feel such impact though. So the identification of the lagged effect is not a trivial issue.

Current statistical techniques, namely Granger causality [2] and transfer functions [3], mostly rely on building a regression model where the independent variables represent the same metric, but collected at some time in the past. For example, assuming that we would like to know the effect of time series x on time series y, with a maximum possible delay of 2 time periods, the following simple regression model could be built:

$$y(t) = a*x(t)+b*x(t-1)+c*x(t-2) \qquad (1)$$

where y(t) is the value of y at time t; x(t), x(t-1), and x(t-2) are the values of x at time t (no delay), at time t-1 (delay of 1), and at time t-2 (delay of 2) respectively; and a, b, and c are the regression coefficients. Note that more complex models can be built, including those where past values of y are used as well (for an autoregressive part).

The limitation with such a model is that it assumes that the delay in the effect is always constant, which is not always the case. Also, for a more general delayed effect such as "between 1 and 3 time periods", one has to guess it and add the corresponding variable in the model being built. Trying all possible combinations of such kind of variable can be computationally expensive.

In this paper, we describe our initial work on a technique for solving such kind of problem. It relies on the identification of peaks in the two time series. Similar peaks across time series are then matched, showing at the same time the delay in the effect. The next section describes this technique with an

(DOI reference number: 10.18293/SEKE2015-042)

example, also comparing it with known statistical techniques. Section III provides information to help choose the right parameters for this technique. The last section provides other examples where we used this technique, as well as a discussion of the future work.

II. Peak analysis

In order to build an approach for identifying lagged effect, an example for which the actual effect could be found in some other way was needed, to confirm its correctness. The following metrics were thus chosen for an initial analysis: number of bugs opened vs. number of bugs closed at each week during software evolution. There should be a lagged effect between them, as many of the open bugs eventually get closed.

We collected such data on JEdit, a Source-Forge project, over a period of 119 weeks. There were 229 bugs found during that period. We also extracted information on when each bug was opened and closed, and the duration between these two events. The distribution of these durations is clearly exponential: 50% of the bugs were fixed within the same week, 10% of them were fixed a week later, 10% of them fixed 2 to 4 weeks later, 10% between 5 and 10 weeks later, and the rest taking more than 10 weeks to be fixed, up to a maximum of 85 weeks.

We first checked our claim that other statistical techniques were not producing valuable results in our situation, by applying them to our data. The best model that could be built was the following (p-value = 0.000):

$$close(t) = 0.173 + 0.596\ open(t) + 0.331\ open(t-10). \quad (2)$$

Although the p-value was small, we had indications that the model was not good for prediction purposes: the R^2 value was only 39% (i.e., only 39% of the variation could be explained by the model), and the residuals were not normally distributed. This is not surprising, considering that only 50% of the bugs have a duration of 0, and less than 1% have a duration of 10. The reason for the difficulty in building a model is due to the fact that development happens in burst, and that the average duration (or lag) is changing over time. Our proposed approach is meant to overcome this issue.

The main idea behind our peak analysis technique is that if there is a lagged effect between two time series, peaks in one time series should match peaks in the other time series, with the lag being the time distance between the corresponding peaks. A peak here is defined as a time interval for which the value of the metric is significantly above its average value.

The first step in our proposed technique is to identify independently the peaks in the two time series. One key point in the definition of peak is the fact that we have to compare data with some average. However, one cannot assume that such average will stay the same over a long period of time. For example here, since development typically occurs in bursts, peaks during those bursts are expected to be higher than the peaks occurring during a period of low activity. Still, we are interested in all peaks, not only the ones during bursts. Some kind of local average is thus needed. In statistics, this is handled through the concept of "moving average" [3]: a new time series is constructed using values corresponding to the average of a given number of consecutive values from the original time series. For example, assuming that we have a time series x, a new time series z corresponding to a moving average of 5 (later referred to as "MA5") is composed of the following values:

$$z(t) = (x(t-2)+x(t-1)+x(t)+x(t+1)+x(t+2))/5 \quad (3)$$

for all t=3 to n-2 (n being the length of the time series x). For the values that cannot be calculated (i.e., at times t = 1, 2, n-1, and n), one can pad with the closest value that can be calculated (i.e., set z(1) and z(2) to the value of z(3), and set z(n-1) and z(n) to the value of z(n-2)).

For peak identification, we actually build two new time series through moving averages: one to smooth the data, to be able to see a trend, and one to represent the local average. For our example here, we used an MA5 to smooth the data, and an MA49 (i.e., moving average where datapoints from t-24 to t+24 are averaged) for the local average. Padding is used to ensure that we have the same number of datapoints for the MA5 and the MA49. Figure 1 shows a plot of these moving averages for the count of opened bugs every week, and Figure 2 shows the one for the counts of closed bugs. Note that for space reasons, we show only an excerpt for weeks 30 to 90. Peaks are clearly appearing, as the intervals when the MA5 line is clearly above the MA49 line. In a more formal way, an interval is a peak only if it goes significantly above the MA49 line in at least one of its points. We use the standard deviation of the MA49 dataset as the threshold for being considered "significantly above the MA49".

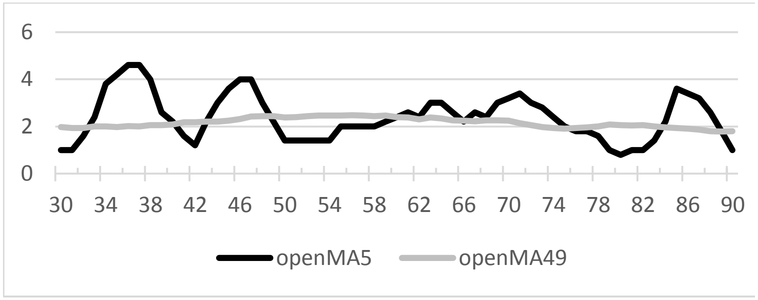

Figure 1. Plot of the MA5 and MA49 on opened bugs, showing peaks.

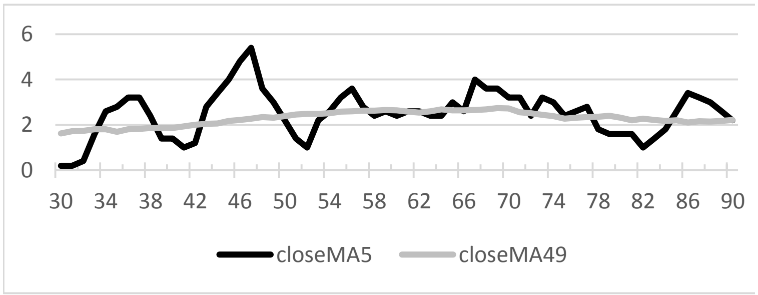

Figure 2. Plot of the MA5 and MA49 on closed bugs, showing peaks.

TABLE I. LOCATION AND SIZE OF PEAKS

Opened bugs			Closed bugs		
ID	***interval***	***size***	***ID***	***interval***	***size***
O1	33--40	12.2	C1	34--38	5.22
O2	43--48	5.98	C2	43--49	11.56
O3	60--75	7.7	C3	54--57	1.88
O4	84--89	5.56	C4	65--77	6.68
			C5	85--89	4.02

Table I lists those peaks, with their time interval and their size. The size is calculated as the sum of the differences between the MA5 and the MA49, for all points in the interval. Note that in the case where two peaks are separated by only one data point falling slightly below the MA49 (by less than the standard deviation on the MA49 dataset), those two peaks are merged into one. The third peak for open bugs is such a merged peak, covering from time 60 to 75, in spite of a dip at time 66.

The second step of our approach is to match the identified peaks in the two time series. The idea is to match peaks of similar size sequentially, trying to match as many peaks as possible. The matches have to be valid though: the start time of the close peak should be no earlier than the start time of the matched open peak, and same for the end times. In our example, the best set of matches is as follows: C1 unmatched, O1 matched to C2, O2 matched with C3, O3 matched to C4, and O4 matched to C5.

We evaluate how good the set of matches is by calculating the Pearson correlation between the corresponding sizes. When a peak is unmatched (e.g., C1 here), we associate it with a peak of size zero. In the example above, the correlation between the vectors [0.0, 12.2, 5.98, 7.7, 5.56] and [5.22, 11.56, 1.88, 6.68, 4.02] is 0.65. This is the best possible correlation for the data above. The algorithm for finding such best set of matches builds all possible sets using a backtracking approach, where the following conditions apply: for all peak Oi matched to a peak Cj, (a) each open peak earlier than Oi is either not matched or matched to a close peak earlier than Cj, and (b) each open peak later than Oi is either not matched or matched to a close peak later than Cj. For each possible set of matches, the correlations is calculated, and the set corresponding to the best correlation is kept.

In the best set of matches identified above, one can see that many of the matched peaks are very close in size (i.e., O1—C2, O3—C4, and O4—C5). However, the peak C1 is quite large for an unmatched peak, and the matched peaks O2—C3 are very different in size. This is why the correlation is quite weak. Looking at the graphs again (Figures 1 and 2) and the sizes of the peaks in Table I, one can see that the peak C1 could actually be somewhat covered by the peak O1, with the remaining of O1 covered by C2 (C2 is large enough to cover both O2 and part of O1). In order to spot these possibilities, the algorithm above can be repeated using half peaks rather than full peaks, in trying to identify the best matches: each peak is cut in half, and considered as its own separate peak, cutting in half the original interval and size. The correlation is still done using the full open peaks though, adding the corresponding half peaks when possible. Table II shows the results for the half-peak analysis. Corresponding peaks are the ones on the same row in the table. The correlation in this case is really high at 0.969 (between the vectors [0.0, 12.2, 5.98, 0.0, 7.7, 5.56] and [5.22/2, 5.22/2+11.56/2, 11.56/2, 1.88, 6.68, 4.02]).

TABLE II. MATCHED PEAKS AND HALF-PEAKS

Opened bugs			Closed bugs		
ID	***interval***	***size***	***ID***	***interval***	***size***
			C1	34--38	5.22
O1	33--40	12.2			
			C2	43--49	11.56
O2	43--48	5.98			
			C3	54--57	1.88
O3	60--75	7.7	C4	65--77	6.68
O4	84--89	5.56	C5	85--89	4.02

We also tried to see if we could detect that one variable does not have an effect on the other. In the case here, we would not expect the count of close bugs to lead to a later effect on the count of open bugs, as very few of the close bugs get re-opened. So we ran our algorithm again, reversing the two time series. In this case, the best correlation was only 0.27 (for the whole 119 weeks of the example, not just the excerpt presented here). The half-peak analysis could improve this to 0.75, but this is still too low to be considered a true effect.

The obvious question now is: does this correspond to what is actually going on with the bugs in those time intervals? The answer is yes. For all of the peaks O2 to O4, between 70% and 82% of the bugs were closed within the interval of their matched peak. In the case of the peak O1, 43% of its bugs were associated to C1, while 23% of them were associated with C2.

We also looked at the average durations of the bugs (i.e., number of weeks between the time they are opened and the time they are closed) within each of the open peaks. These were 9.07, 10.65, 3.88, and 2.41 for peaks O1 to O4 respectively. This seems in line with our initial findings: peak O1 has a much higher average duration due to its match to a peak that is at a distance of 9 to 10 weeks. It still contains a large percentage of bugs fixed almost immediately (i.e., duration of 0 or 1). The peak O3 is matched to a peak that is at a distance of 2 to 5 weeks, matching its average duration. The average duration for the peak O4 is slightly larger than its distance to its matched peak, but this peak does contain a much larger percentage of bugs fixed immediately than the entire set of bugs (71% vs. 51%). For the peak O2, its average duration does not seem to match its distance to its matched peak. This is due to the fact that it contains a much larger percentage of bugs that are fixed within a very long time frame, with a large spread in duration. Actually, we could have seen the problem in the first place in our analysis, if we had considered the second best set of matches: if we match O2 to both C2 and C3, the correlation is still really high at 0.953. And, the distance between peaks O2 and C3 is approximately 10, which does correspond to the average duration.

As one can see, such an analysis can show how the distribution of bug durations changed over time. Using such information, one could try to identify what exactly happened at those times to cause such differences (e.g., perhaps the kind of maintenance – corrective vs. perfective – was different). We did not have enough knowledge of this particular software evolution (of JEdit) to perform such investigation.

III. Choice of Values for Moving Averages

In the technique as presented above, two moving averages are used (one to smooth the data and one to act as a local average) for identifying peaks. In the example provided, moving averages of 5 and 49 were used. Other numbers can be used as well, depending on what exactly one would like to find out.

The analysis of changes in the lagged effect over time (as mostly performed above) does not help identify the general lag in the overall effect. Increasing the size of the moving average used can help with this. For example, using the context and data above but a moving average of 9 rather than 5, we get 3 peaks that are matched to 3 other peaks of roughly the same size. The apparent lag in this case is between 0 and 3, which corresponds to the duration of 70% of the bugs. By doing this tough, we lose the possibility of identifying the times when such lag was different. On the other hand, if we want to be even more specific in the lags and the differences at particular times, a smaller moving average (e.g., MA3) can be used.

By increasing the size of the moving average, the length of the intervals for the peaks found increases, and the number of such peaks is reduced. Then, when a match is found, the peaks tend to be of more similar sizes. However, because of the longer interval, there could be more bugs that have a much higher duration than what would be seen when looking at the distance between the matched peaks. For example, if matched peaks are located in the interval [30..45] and [31..46], although the apparent delay is just 1, there could still be many bugs with a much higher duration (e.g., a bug being opened at time 30 but closed at time 45).

For the size of the moving average used as the local average (e.g., the MA49 used in the previous section), our experience shows that an appropriate number should be 5 to 10 times larger than the moving average used to smooth the data.

IV. Validation and Discussions

In Section II above, we have shown how our technique works on a subset of the data we had from the open source software JEdit. It should be noted that the technique was successful for the entire 119 weeks of data that we had, which included an extra matched peak prior to week 30, and an extra matched peak after week 90 (not displayed above).

We validated our technique on a second open source system: MinGW (another SourceForge project). We extracted the same kind of data as described in Section II above, over a period of 175 weeks. There were 203 bugs found during that period. Through our technique (using an MA5 vs. an MA49 as described above), we could identify 11 peaks for opened bugs and 11 peaks for closed bugs. Those peaks were very diverse, with interval lengths ranging from 2 to 14 weeks, and with sizes ranging from 0.2 to 9.5. The duration of the bugs was somewhat longer than the ones found in JEdit, with the following distribution: 27% of the bugs were fixed within the same week, 26% of them were fixed a week later, 13% of them fixed 2 to 4 weeks later, 14% between 5 and 10 weeks later, and the rest taking more than 10 weeks to be fixed, up to a maximum of 123 weeks. We successfully matched the related peaks with a correlation of 0.84 (0.90 when improving it using half-peaks). In almost all cases, the majority of the bugs (62% to 100%) opened within a given peak were closed within the matching peak. There was one exception, but this was with a relatively small peak, matched to a much larger peak (i.e., with very different sizes).

From that system (MinGW), we also analyzed the time lag between the number of commits per week and the number of opened bugs per week (i.e., how long does it take in general to find bugs after modifications are made). We saw that such lag was approximately 8 to 9 weeks, with one exception where the lag was only 3 to 5 weeks. That time corresponded with an increase in new features developed. Such kind of information can be useful in planning when there will be an increase in demand for fixing bugs.

We also used our technique to confirm previous results in another (unrelated) project, where we have shown that making many highly-dispersed changes in a file was increasing the risk of finding a bug in that file within three months after the change [4]. This previous work was looking at individual files going through a sudden burst of changes, characterizing those changes and predicting the bugginess of the file based on similarities with past cases. Here, we looked at the proportion of the file commits performed every week, for all files rather than individual files, that were implementing highly-dispersed changes. We compared this with the number of bugs detected every week. And indeed, the peaks in these two time series were matching, with a typical lag between 2 and 7 weeks. This confirmed our previous results, with even more precise information about the lag. We repeated the work on other types of changes (e.g., small local change, massive local change), but we could not see a match in the peaks. This supported our previous findings too, that other types of changes were either not affecting the bugginess of the file, or were affecting it only when the file was large. Not only did we confirm previous results here, but such analysis could help building a better bug predictor.

As future work, further validation is clearly required. At this point, we tried our approach on bug data for only two systems: JEdit and MinGW. We need to try it out on more (various) systems, over a longer period of time, and on other kinds of metrics that could be analyzed this way. Improvement to the underlying algorithm is also necessary in order to make it more efficient, and practical for larger inputs. Finally, we are interested in applying this kind of technique to areas other than software engineering.

References

[1] M. D'Ambros, M. Lanza, R. Robbes, "An Extensive Comparison of Bug Prediction Approaches", Proc. of the 7th IEEE Working Conf. on Mining Software Repositories, Cape Town, South Africa, May 2010, pp. 31-41.

[2] C.W.J. Granger, "Testing for causality: A personal viewpoint", Journal of Economic Dynamics and Control, vol. 2, pp. 329-352, 1980.

[3] D.C. Montgomery, C.L. Jennings, and M. Kulahci, Introduction to Time Series Analysis and Forecasting, Wiley, 2007.

[4] J. Tassé, "Using code change types in an analogy-based classifier for short-term defect prediction", 9th Int. Conf. on Predictive Models in Software Engineering, Article No. 5, Baltimore, Maryland, Oct. 2013.