Website Changes and User Behavior: Using Panjiva Data to Examine Code Changes

John Wang
14.27 Final Paper

November 29, 2012

Abstract

blah

Contents

0.1 Dataset Explanation

The proprietary Panjiva Dataset comes from the back-end databases collected by Panjiva, Inc. Panjiva's website `http://www.panjiva.com` acts as a medium for buyers and suppliers of manufactured goods. The site provides a communication platform so that bulk buyers of a particular good can search and obtain unbiased information on factories and suppliers of that good. These two parties can then communicate and send messages over Panjiva's interface, attempting to strike a deal.

Panjiva's competitive advantage rests in its ability to parse government import and export data in order to obtain unbiased information about suppliers. Panjiva determines a supplier reliability score and also provides recent history of a supplier's shipments, and allows buyers to search and aggregate this information easily. Most firms that use Panjiva are large to medium size buyers of components. For example, a department store would use Panjiva to search for suppliers of shirts or clothing, or a home improvement store would search for suppliers of socket wrenches. In addition, Panjiva provides data on trends in global manufacturing and shipping by leveraging the government data it already mines for individual supplier information.

The dataset used in this paper comes from the event and activity logs of Panjiva's website. Each time a user performs some significant event or activity on the Panjiva website, an entry will be created in either the event or activity log. If the user has a registered account with Panjiva, the action will be recorded in the activity logs. All activities, regardless of whether a user has an account, are recorded in the event logs. All nontrivial features of the website, including supplier search, U.S. import and export search, and profiles views, are accounted for and stored in a SQL database.

The data in the event and activity logs are organized so that one can trace the exact user or subscribed account for which the entry. In particular, the logs contain information on the ip address of the user, the time the activity was performed, the webpage the activity occurred on, and extra data depending on the type of activity performed.

The enormous quantity and granularity of this data enable the analysis of user-level interactions. The event logs contain about 124 million entries while the activity logs contain about 13 million entries. Moreover, the data in each of these logs extends for multiple years, allowing one to analyze the growth of Panjiva as a company and the effect of different changes to the website.

0.1.1 Summary Statistics

Summary statistics providing an overview of Panjiva's business and website are provided in table 1. Panjiva was incorporated in 2008 and has since developed a customer base composed mostly of buyers of manufactured goods. The website provides free services (a limited number of searches) for free users, and provides many more services to subscribers (there are multiple levels of subscription).

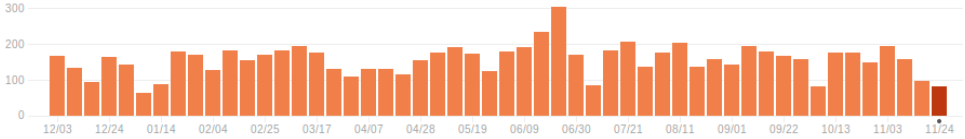
Table 1: Panjiva Overview

Total Users	121,653
Subscribing Users	2,985
Monthly Site Visits	903,426
Monthly Unique Visitors	762,723
Average Pages per Visit	1.99
Average Visit Duration	1 min 18 sec

Commits can be thought of as changes to the code. In each commit, a software developer will introduce new code or delete old code which will then be launched to the production website. Typically a single developer at Panjiva will commit a couple of times throughout the workday. Figure 1 shows a histogram of the number of commits throughout a 52 week period ending on 11/24/2012. Commit activity varies depending on the season and the week. In particular, commits during the summer spike upwards and there will be one or two weeks each quarter when commits fall to low levels. The second phenomenon is due to the fact that Panjiva holds a quarterly retreat where engineers reflect upon the work done over the quarter. Typically the amount of code written decreases during these weeks.

Table 2 shows statistics on a snapshot of the commit repository made on 11/25/2012. The table shows an enormous number of changes throughout the history of the website. In addition, the 1.3 million lines

Figure 1: 52-Week Commit Activity



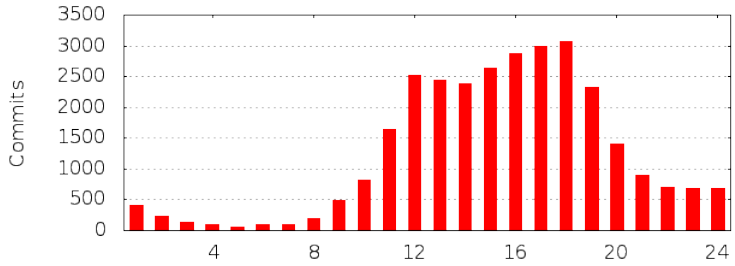
of code show that one can use the variation in the lines of code that a commit changes in order to better understand how each commit affects user behavior.

Table 2: Overall Commit Statistics - 11/25/2012

Active Days (at least 1 commit)	1,983
Total Current Files	20,901
Total Lines of Code	1,313,235
Total Lines of Code Added	3,989,295
Total Lines of Code Removed	2,676,060
Total Commits	29,924
Total Authors/Developers	33

Examining commits in more detail, one can see that the majority of commits occur during working hours (from 11am to 6pm EST). Moreover, each day is fairly regular in the number of commits that occur. Commits are at a high level throughout the work day, and fall off to a low, relatively constant level throughout the night. Figure 2 shows the times of day during which commits happen the most frequently.

Figure 2: Commit Frequency by Time of Day



Finally, note that almost no commits occur on weekends. Only 5% of the commits happen on either Saturday or Sunday.

0.2 Macro-Level Results

This section examines how code changes affect the total amount of user activity across the entire website (macro-level effects). In particular, this section analyzes the change in a number of metrics of user activity controlling for the amount of code changed in a particular day. The regression specification used will the as follows:

$$y_t^i = c_0 + \bar{\gamma}^T \bar{M}_t + \bar{\beta}^T \bar{\chi}_t + \epsilon_t \tag{1}$$

Where t indexes day, y_t^i corresponds to the ith metric of user activity on day t, $\bar{M}_t$ corresponds to a vector of covariates that represent changes in the code, $\bar{\chi}_t$ is a vector of controls, and ϵ_t is an error term. We will use a number of different metrics for user activity and the specification will be run with each of them.

The first two metrics for user actions are the total number of entries in the activity or event logs for a particular time period. Recall that event logs are database records for actions performed by anyone, while activity logs are records for only those users who have a registered account with Panjiva.

Another set of metrics are the total number of distinct users performing either activities or events in a given time period. Finally, one might also use the average number of actions per user, whether action is defined using records from activity or event logs. One would expect the each of these metrics to measure a different aspect of user behavior.

0.2.1 Daily Effects of Code Changes

Table 3 displays the results from using the total action count as a metric for user activity. The regressions in table 3 attempt to find the effect of different changes in the code base on different measures of user activity. A commit can be measured by the number of files it changes, or the number of lines of code it changes. Changes in lines of code can be either insertions or deletions. Each of these measures is percentilized (sorted rank is divided by the total number of observations). The regressions below are performed on aggregated statistics per day, with a dummy variable equal to 1 on the weekends and 0 otherwise.

The table shows the effect of commits on same day user activity. In general, changes to the code have minor and insignificant effects on metrics derived from the event logs. There are only significant effects for the eventlogcount. However, code changes have a stronger effect on metrics derived from the activity logs. Notice that for activitylogcount, the insertionspercentile variable is significant and the fileschangedpercentile and deletionspercentile are within the 10% significance level. Moreover, fileschangedpercentile has a significant effect on the average number of activities per user. The number of files changed and number of insertions are the only significant regressors, and they have opposing signs.

Increasing the number of files changed tends to decrease user activity, while increasing the number of code insertions tends to increase user activity. Although this result seems perplexing, there are a number of possible explanations. First, it is possible that the commits which change fewer files tend to be higher quality commits. Thus, if a commit touches a large number of files, there is a higher likelihood that this commit would introduce a bug or contain low quality changes. However, it seems that introducing new code, instead of deleting code, has a positive effect on user activity, controlling for the number of files changed.

Another, though possibly less satisfying explanation, is that code changes hurt average user activity in general. Since average user activity provides a measure of the organic growth of the website, it may be a better measure for the effects of code changes. Note that increasing total user count could result from a large number of new distinct user performing one or two actions each on the website. Although this could be construed as an increase in activity, each user could leave quickly, which would negatively affect long term traffic. Thus, one can think of the negative coefficient on the fileschangedpercentile variable as decreasing the long-term user activity with more code changes.

However, the second explanation seems unlikely because of the fact that code changes are constantly occuring. The regression is measuring the marginal effect of commits. Each additional file changed could provide a negative effect due to diminishing returns. Thus, one could have a negative coefficient on fileschangedpercentile even when increasing the number of files changed at very low levels (i.e. from 0 to 1) would increase user activity.

Also, notice that commits have no significant effect on the total number of distinct users. This is unsurprising since it is likely that it takes a significant period of time to increase the number of distinct users. Changes to the website will probably not spill over to other users during short time spans (like a day). Users do not have the time yet to share their new experiences on the site, so each user can be thought of as independent of others during these short time periods. Therefore, it is unsurprising that code changes fail to increase the total number of distinct users on the website.

0.2.2 Lagged Effects of Code Changes

Note that the analysis of the previous section uses the implicit assumption that for a given day t, only the code changes on day t influence user actions on day t. The previous regression assumes first that there is no lag between a code change and the change in user activity resulting from the code change, and second

Table 3: Effect of Commits on User Activity

	(1) activitylogcount	(2) eventlogcount	(3) activitylog distinctusercount	(4) eventlog distinctusercount	(5) avguseractivity	(6) avguserevents
fileschangedpercentile	-5530.7	-60589.4*	460.5	492.7	-10.40*	-148.6
	(-1.91)	(-2.26)	(0.65)	(0.71)	(-2.01)	(-1.52)
insertionspercentile	4868.8*	47053.7*	284.2	118.6	2.817	79.92
	(2.12)	(2.22)	(0.51)	(0.22)	(0.69)	(1.03)
deletionspercentile	2778.9	29970.6	-192.2	-184.4	4.670	49.47
	(1.29)	(1.50)	(-0.36)	(-0.36)	(1.21)	(0.68)
weekend	-14708.0***	-79769.8***	-1060.6***	-816.2***	-0.297	257.0***
	(-16.48)	(-9.65)	(-4.86)	(-3.82)	(-0.19)	(8.49)
_cons	22396.6***	224482.8***	1635.7***	1395.9***	25.00***	340.6***
	(25.05)	(27.12)	(7.49)	(6.52)	(15.58)	(11.24)
N	474	475	474	475	474	475

t statistics in parentheses

* $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

The table provides regression estimates of how different measures of user actions vary with changes in the codebase. The table was created by aggregated data from the Panjiva activity and event logs over the period from 7/14/2011 to 11/24/2012. Variables are broken apart by day, so that counts are per day. Activity logs record information about users with registered accounts, while event logs record information about all users regardless of whether they have registered with the Panjiva.

Activitylogcount and eventlogcount aggregate the total number of records seen per day in the activity and event logs respectively. Activitylogdistincusercount and eventlogdistinctusercount aggregate the number of distinct users that have performed an action in either the activity or event logs respectively. Finally, avguseractivity and avguserevents divides the total number of records by the total number of distinct users per day to arrive at an average number of actions per day.

Weekend is a dummy variable equal to 1 if the day is a Saturday or Sunday and 0 otherwise. The other covariates are percentilized by sorting each day and taking the rank and dividing by the total number of days. Files changed, insertions and deletions are obtained from commit history and are aggregated over each day. Insertions and deletions are the total number of lines of code inserted or deleted, respectively.

that only a single day's code changes affects the user activity on day t. Both of these assumptions are simplifications, and this section will relax the first one.

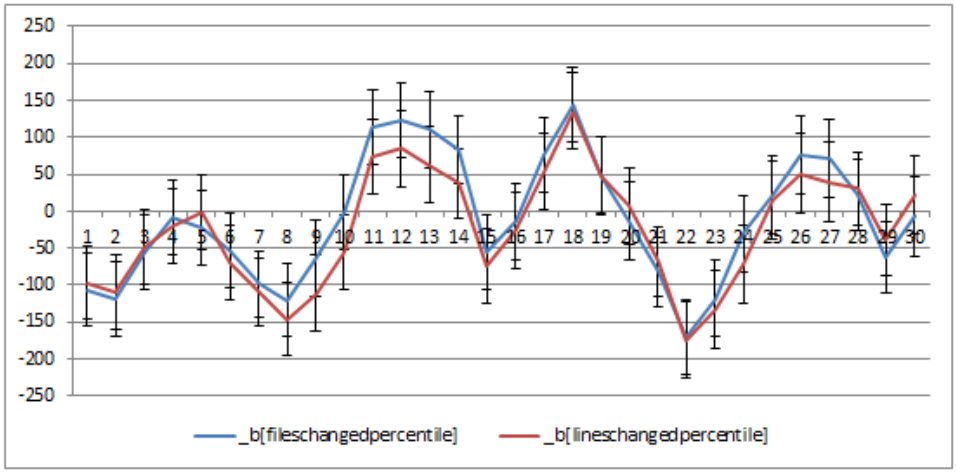
The original specification of the regression will now depend upon y^i_{tx}, where x is a number of days of lag. Thus, if $x = 5$, then y^i_{tx} will correspond to the y^i_t which was five days ahead in the original specification. The specification will be run for all x in some range $[0, R]$ to identify whether code changes have a lagged effect upon the changes in user activity. Figure 3 shows the change in the coefficients of fileschangedpercentile (in red) and lineschangedpercentile (in blue) with lags of $x \in [1, 30]$ when the regressor is average events per user from the event logs. Figure 4 provides the same information for the regressor of average activities per user from the activity logs.

The coefficient plotted in these figures is the γ coefficient from the following specification (where x varies from 1 to 30):

$$y^i_{tx} = c_0 + \gamma M_t + \beta weekend_t + \epsilon_t \tag{2}$$

Like the previous specification, M_t is a measure of code change in day t (either fileschangedpercentile or lineschangedpercentile) and $weekend_t$ is a dummy variable equal to 1 if t represents a day which is either Saturday or Sunday. Each figure plots the γ coefficient (on the y-axis) from the above regression with different time lags ranging from 0 to 30 (on the x-axis).

Figure 3: γ Coefficients with Varying Lags, Regressed on Average Event Logs per User



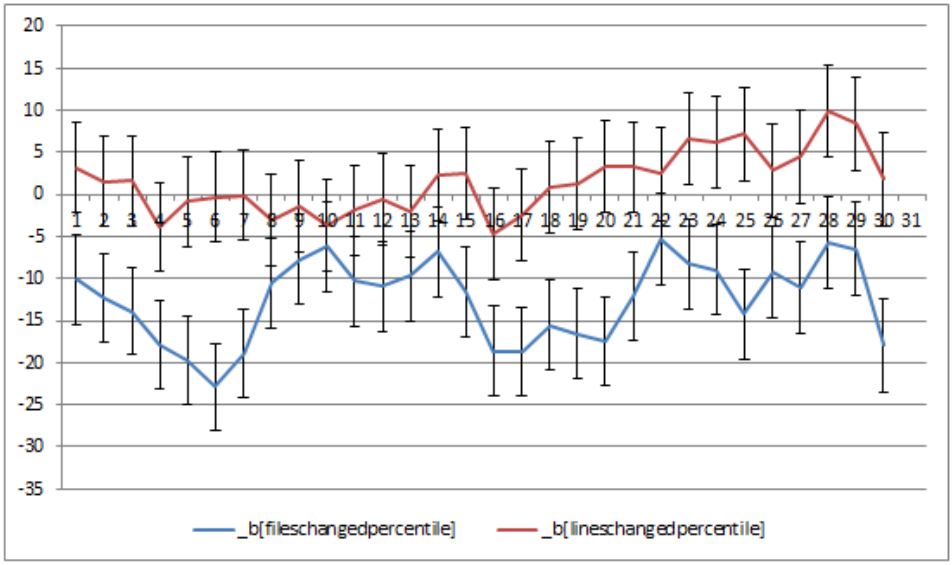
Examining figure 3, it is difficult to ascertain the actual effect of the change in time lags (note that error bars correspond to standard errors). Both covariates representing code changes, fileschangedpercentile and lineschangedpercentile, have very similar coefficients across different time lags. Small time lags less than 10 result in negative and significant coefficients for both fileschangedpercentile and lineschangedpercentile, while larger time lags tend to have positive and significant coefficients. The second trend, however, is bucked by a couple of negative coefficients when $x \in [21, 24]$.

The amount of noise inherent in the event logs probably makes the graph difficult, if not impossible, to interpret. One could hypothesize that the shape of the curve in figure 3 suggests that code changes have differential effects depending on the amount of time lag one accounts for. However, this explanation seems week since event logs contain records of all possible users, making it difficult to interpret overall changes in the trend without decomposing into well-defined groups of users. Second, figure 4 shows that the same results performed on the action log records provides substantially more elucidating results.

Figure 4 shows large and consistent negative coefficients on fileschangedpercentile with all different ranges of lag. The figure also shows that the coefficient on lineschangedpercentile is not significantly different from zero for almost all lags, suggesting that it has little effect in predicting user activity.

However, the number of files changed per day has a significant and negative coefficient throughout all time lags, reinforcing the results from the previous section that the more files changed on a particular day, the lower user activity tends to be. This result stays consistent across time, suggesting that lower user

Figure 4: γ Coefficients with Varying Lags, Regressed on Average Activity Logs per User



activity begins early and persists for a number of days. This provides more evidence that increasing the number of files changed per day, given the current number of files changed, will decrease user average user activity from users who are registered on Panjiva.

Figure 5: γ_0 and γ_1 Coefficients with Varying Lags. Regressed on Average Activity Logs per User, the graph displays the coefficients on insertionspercentile and deletionspercentile as time lags range from 0 to 30.

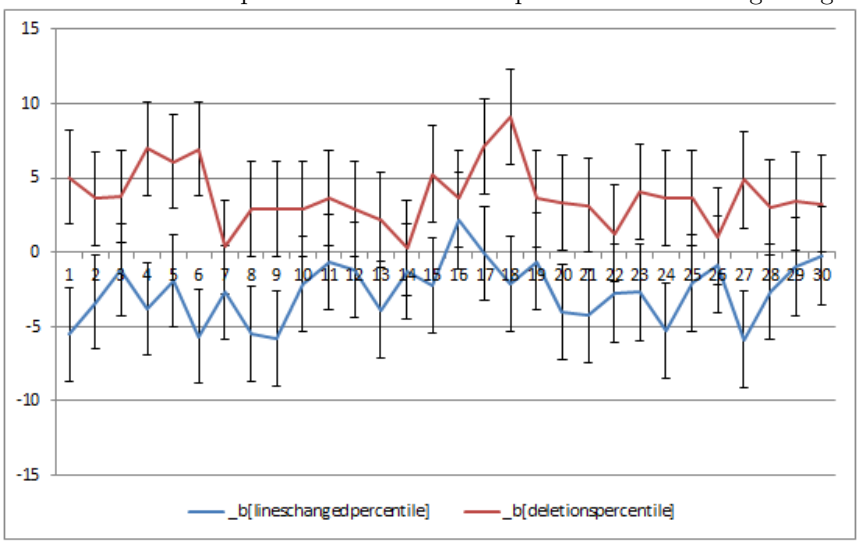


Figure 5 examines the change in average user activity due to insertions and deletions of code. This graph makes the behavior of coefficients on insertionspercentile and deletionspercentile from the original regressions more clear. The figure displays the γ_0 and γ_1 coefficients from the following regression:

$$avguseractivity_{tx} = c_0 + \gamma_0 insertionspercentile_t + \gamma_1 deletionspercentile_t + \beta weekend_t + \epsilon_t \quad (3)$$

The graph remains consistent across different levels of lags-insertionspercentile's coefficient tends to be negative while deletionspercentile's coefficient tends to be positive. Although there exist multiple lags for which either γ_0 or γ_1 is not significantly different from 0, it is never the case that the γ_0 and γ_1 coefficients

switch sign. This implies that the actual effect of increasing the insertion percentile is likely correlated with lower average user activity while increasing the deletion percentile does the opposite.

This finding seems to imply that removing code tends to be better than adding code. Although this hypothesis was reached on macro-level data, it is still an interesting idea. Perhaps removing code is correlated with making better code, and inserting new code usually happens with a new, possibly buggy feature being added. If this is the case, then it seems that users respond better to clean, non-buggy code. To investigate this further, regressions are performed on a more granular level.

0.3 Micro-Level Results

Although the macro-level regressions provided some insight into how code changes affect user activity, they really could not provide anything more than correlations. Complex factors could have been governing the relationship between code changes and user activity, and only weekend dummies were included in the regressions. Thus, the macro-level results must be taken with a grain of salt since the correlations observed could be brought through completely different mechanisms than the ones suggested.

However, the micro-level dataset can be much more powerful in its results. The dataset used in this section exclusively uses the activity logs, and exploits the fact that each recorded activity comes with a unique user identification tag and timestamp. Therefore, it is possible to know exactly when a user was visiting a given page.

Even better, most of the code changes made throughout the day can be thought of as exogeneous shocks. This arises from the fact that almost none of the releases made by developers are announced to the user base beforehand. Even if the changes are announced, few users would be notified and aware of the change because Panjiva typically advertises such changes through its blog. Less than 1% of Panjiva's total pageviews come from its blog, so it is safe to assume that most users are unaware of any planned changes to the website. Since these code changes can be thought of as exogeneous with a high degree of accuracy, one can develop an empirical approach that examines the shock induced by each code change.

0.3.1 Search Controller

The search page has always been Panjiva's most trafficked page. Possibly due to this, it has received a number of code changes, both in terms of back-end structure and also in terms of the user interface. There is, correspondingly a large amount of variation in the number of lines of code changed and the times at which code was changed on the search page. To exploit this variation, this section will examine user activity before and after a code change.

In particular, this section limits the activity log dataset to only users who have seen the search page both two days before and after a particular change. Thus, for each code change in the search page, the users that had recently seen the unchanged page and also the changed page were examined. Any users seeing the search page over two days before the change were dropped, as were users who saw the search page over two days after the change. User activity was measured with the $num_views_day_after$ variable, which corresponds to the number of views of the search page in the day after the current view of the page. Note that day after is defined as 86,400 seconds after the original view occurred.

Since the dataset was confined to only users who saw both the original website and the changed website, one can take the difference between the number of views before and after the change as the effect of the change (assuming that commits are exogeneous). Another refinement made to the dataset is that only commits to the search controller were examined.[1] This choice was made because a significant portion of the user interface code is antiquated and left in the code-base. Therefore, large changes to the user interface code could spuriously affect parts of the code that no longer are seen on the production website.

The time period from 7/14/2011 until 11/25/2012 was examined (to confine the dataset to more recent changes). A total of 294 changes occurred during this time, ranging from small one line changes to larger

[1] A controller in a Model-View-Controller web framework is typically the piece of the code that manages the data and sends it along to the user interface. For Panjiva, almost all of the heavy-lifting is done in the controller (this includes data processing and the search algorithm itself).

changes in hundreds or thousands of lines. The following regression specification was used:

$$num\_views\_day\_after_{it} = c_0 + \beta_0 \mu_{it} + \beta_1 hour\_dummies_{it} + \epsilon_{it} \tag{4}$$

Where i indexes the ith commit and t indexes the tth view in the 4 day window surrounding the ith commit. Here μ_{it} is either a dummy variable $after\_commit$ which is equal to 1 if the view of the search page occurred after the commit and 0 otherwsie, or it is equal to $time\_from\_event$ which is the number of seconds since the commit. Therefore, if a view of the search controller occured at time T_1 and the commit occurred at time T_2, then $time\_from\_event = T_1 - T_2$.

Table 4: Search Controller Micro-Level Results

	Independent Variable: num_views_day_later			
	(1)	(2)	(3)	(4)
after_commit	5.541***		3.378***	
	(17.91)		(11.20)	
time_from_event		0.0000737***		0.000110***
		(43.76)		(66.60)
created_at_hour dummies?	No	No	Yes	Yes
_cons	156.9***	159.9***	138.3***	139.7***
	(731.32)	(1033.47)	(127.71)	(130.50)
N	2345617	2345617	2345617	2345617

t statistics in parentheses

* $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$