

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

John Wang
14.27 Final Paper

December 8, 2012

Abstract

Most software developers implicitly believe that change tends to improve a website. However, the story seems far more complicated. This paper examines how code changes affect user behavior on the internet by analyzing a particular website, <http://www.panjiva.com> and its database of activity logs. The paper identifies some factors which affect how code changes influence user activity levels and finds patterns in how users respond to change. Users tend to increase their repeat usage when code changes are of high quality and when the changes affect portions of a website which users already frequent. Users respond negatively to changes in areas of the website with low traffic. A massive volume of code change, in general, does not necessarily lead to higher user activity on a website and in fact may negatively impact the user experience.

Contents

1 Introduction

Silicon Valley has historically been built on constant change. Indeed, one of the tenets of technology companies and web-based startups is the idea of change. For example, Facebook’s official motto is “Move fast and break things.” However, few people have examined the effects of website and product changes on user activity. Do users tend to respond favorably to website changes? Moreover, how do users react to different types of changes? Can different characteristics of users change how users respond to certain changes?

All of these questions are crucial and allow developers to create better websites and tools for their users. This paper examines the effects of code changes on user activity. In particular, the paper focuses on the short term effects of everyday changes to a website. The paper exploits a unique dataset and examines how short-term user activity (both overall on the entire website and on a more granular, per-user level) are affected by code changes. The unique dataset comes from Panjiva, Inc., which provided the author access to its entire database.

2 Dataset Explanation

The proprietary dataset used in this paper 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 corresponds to. 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.

2.1 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 Registered 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.

Figure 1: 52-Week Commit Activity



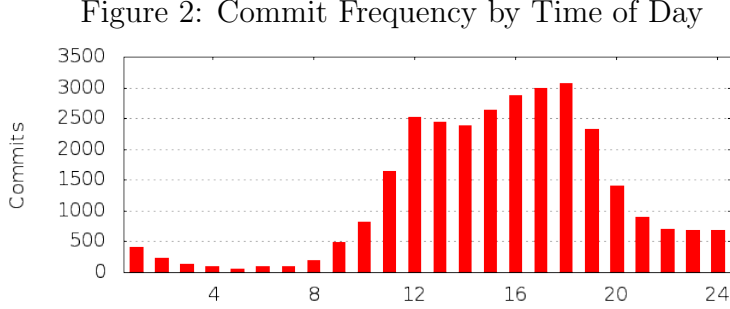
Table 2 shows statistics on a snapshot of the commit repository made on 11/25/2012. The table shows an enormous number of changes throughout the history of the website. In addition, the 1.3 million lines of code show that one can use the variation in the lines of code that a commit changes in order to 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.



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

3 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 be as follows:

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

Where t indexes day, y_t^i corresponds to the i th 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 each of these metrics to measure a different aspect of user behavior.

3.1 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 (see the data appendix for details). The regressions below are performed on aggregated statistics per day, with a dummy variable equal to 1 on the weekends and 0 otherwise.

Table 3: Effect of Commits on User Activity

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

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 occurring. 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.

3.2 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 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_{tx}^i , where x is a number of days of lag. Thus, if $x = 5$, then y_{tx}^i will correspond to the y_t^i 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.

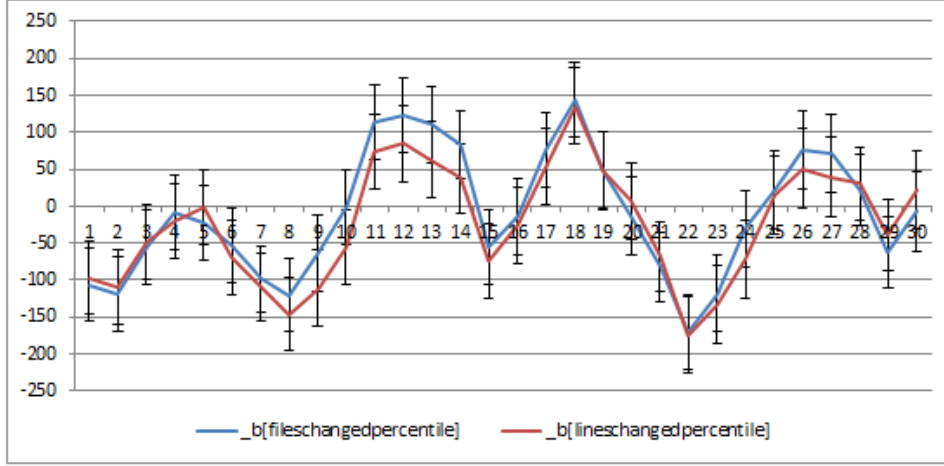
Figure 4 provides the same information for the regressor of average activities per user from the activity logs.

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

$$y_{tx}^i = c_0 + \gamma M_t + \beta \text{weekend}_t + \epsilon_t \quad (2)$$

Like the previous specification, M_t is a measure of code change in day t (either `fileschangedpercentile` or `lineschangedpercentile`) and `weekendt` 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



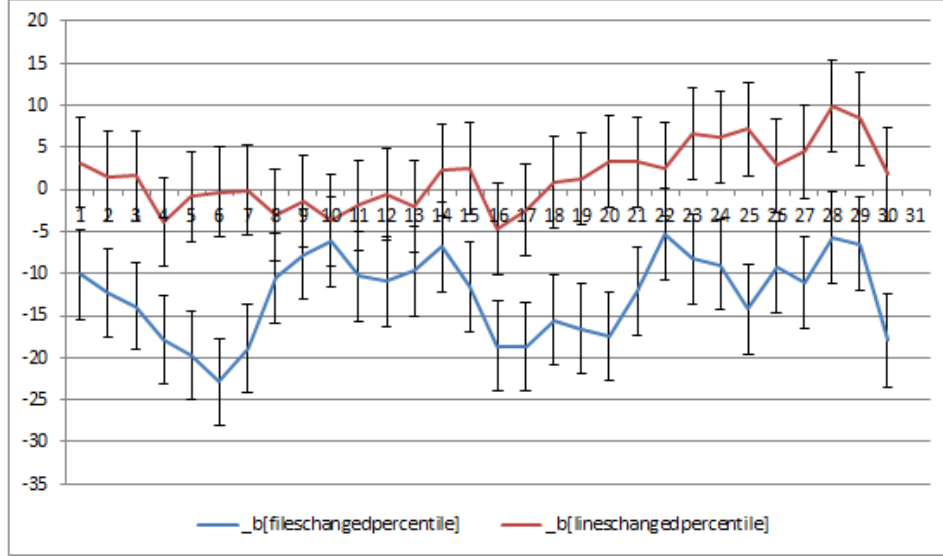
Examining figure 3, it is difficult to ascertain the actual effect of the change in time lags (note that error bars correspond to standard errors). Both covariates representing code changes, `fileschangedpercentile` and `lineschangedpercentile`, have very similar coefficients across different time lags. 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 weak 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 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

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



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.

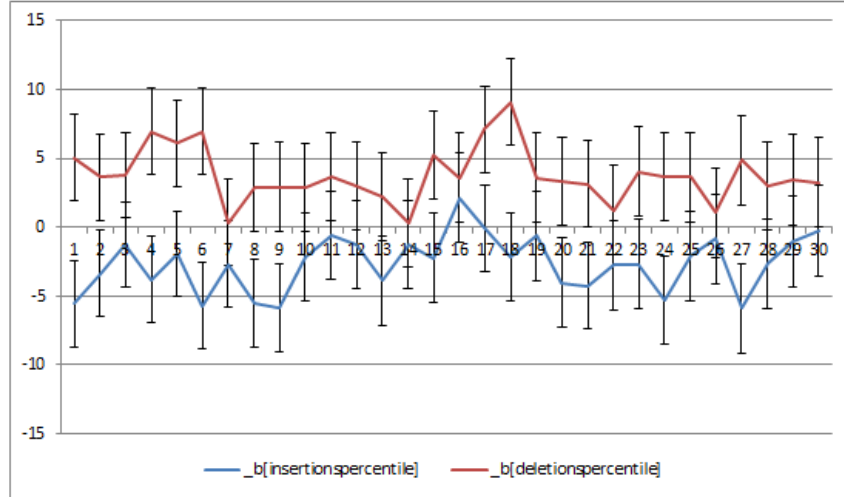


Figure 5 examines the change in average user activity due to insertions and deletions of code. This graph makes the behavior of coefficients on insertionspercentile and deletionspercentile from the original regressions more clear. The figure displays the γ_0 and γ_1 coefficients from the following regression:

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

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.

4 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 exogenous 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 exogenous with a high degree of accuracy, one can develop an empirical approach that examines the shock induced by each code change.

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

$$num_views_day_after_{it} = c_0 + \beta_0 \mu_{it} + \beta_1 hour_dummies_{it} + \epsilon_{it} \quad (4)$$

Where i indexes the i th commit and t indexes the t th view in the 4 day window surrounding the i th 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 otherwise, 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 occurred 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				
	Dependent Variable: num_views_day_later			
	(1)	(2)	(3)	(4)
after_commit	5.541*** (17.91)		3.378*** (11.20)	
time_from_event		0.0000737*** (43.76)		0.000110*** (66.60)
created_at_hour dummies?	No	No	Yes	Yes
_cons	156.9*** (731.32)	159.9*** (1033.47)	138.3*** (127.71)	139.7*** (130.50)
N	2345617	2345617	2345617	2345617

t statistics in parentheses

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

Regression using micro-level dataset. Confined only to users who saw the search page both before commit i and after commit i inside of a 4 day window surrounding the time of the commit. The num_views_day_later variable contains the number of views of the search page for each user one day after the record was created. Commits are confined only to changes in the search controller.

There are about 2.3 million observations in this refined dataset and table 4 shows the regression results of the specification defined above. The table shows that actions recorded after the commit occurs have a significantly higher number of views. Moreover, the positive coefficient on the time_from_event variable shows that as one gets further away from the commit

¹A 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).

time, the number of views of the search page increases. Thus, the number of views that occur after the commit is greater than the number of views before the commit (since `time_from_event` is negative before a commit and positive afterwards). The coefficients on `after_commit` and `time_from_event` are strongly statistically significant (much higher than normal levels), which suggests that commits have a large positive effect on user activity.

The statistical significance remains even when hour dummies are included (a set of 24 dummy variables where `created_at_hour_i` = 1 if the activity record was created in the i th hour of the day and 0 otherwise). This provides strong evidence that code changes in the search controller significantly affect user behavior. Since commits are most likely exogenous, user activity seems to increase significantly due to the changes in the search controller.

4.2 Insertions and Deletions

However, it is still unclear from these regressions why user activity increases. To examine this in more detail, a regression with the following specification was conducted:

$$\begin{aligned} num_views_day_later_{it} = & \beta_0 after_commit_{it} + \beta_1 insertionspercentile_{it} + \beta_2 deletionspercentile_{it} \\ & + \beta_3 insertionspercentile_{it} * after_commit_{it} \\ & + \beta_4 deletionspercentile_{it} * after_commit_{it} \\ & + \beta_5 hour_dummies_{it} + c_0 + \epsilon_{it} \end{aligned} \quad (5)$$

Here, the specification uses interaction terms between the `after_commit` dummy variable and the insertion and deletion percentile variables. Using the interaction coefficients, one can determine the additive effect of insertion an extra line of code on the change in the number of views after the commit. Looking on the β_3 and β_4 coefficients in the above specification will tell one how much additive effect on the number of views there is when a commit increases the insertion or deletion percentile (respectively) by 1 after a commit. In essence, it measures the effect of inserting or deleting lines of code on how users will respond to a change.

Table 5 displays the results of the above specification with and without the hour dummies, and also with different sets of controllers. The regressions in columns 1 and 2 of table 5 use observations confined to users who saw changes in the search controller. Columns 3 and 4 use the same specification, but include observations from seven other controllers as well. These controllers are the communication, customs, my_panjiva, profile, project, us_exports, and us_imports controllers. Each of these controllers corresponds to a different landing page on the Panjiva website.

Including all of the controllers almost doubles the number of observations available at 3.8 million. However, the results with different controllers mirror the results found just in the search controller. In particular, the interaction term coefficients (β_3 and β_4 from the specification) tend to be similar across all the regressions. More insertions in the codebase increase the positive effect of a code change on user activity. One sees that β_3 , which corresponds to the coefficient on the interaction between an increase in the insertions percentile and the `after_commit` variable, is large, positive, and highly statistically significant.

This implies that insertions in the code tend to increase the number of views a user will have on a particular page p of the site, with respect to how much that user viewed p before the code change. Deletions in the code tend to have the opposite effect, though this seems to depend on the controller affect. Although the negativity of β_4 is highly statistically significant on only the search controller, it loses its significance once all controllers are included in the regression in columns 3 and 4. However, one can say that deletions in the code, at least in the search controller, have a significant negative effect on the number of pages a user will view in the future.

Table 5: Types of Commits and User Responses to Code Changes

	(1)	(2)	(3)	(4)
	num_views_day_later	num_views_day_later	num_views_day_later	num_views_day_later
after_commit	-10.86*** (-16.90)	-13.79*** (-22.03)	-325.5*** (-154.63)	-305.9*** (-149.28)
insertionspercentile	-50.97*** (-54.12)	-50.76*** (-55.34)	-650.6*** (-218.09)	-609.5*** (-209.88)
deletionspercentile	66.31*** (70.67)	63.35*** (69.34)	251.5*** (84.73)	230.8*** (79.94)
insertionspercentile *	57.00*** (41.21)	58.05*** (43.09)	393.0*** (89.07)	365.1*** (85.01)
deletionspercentile *	-24.16*** (-17.65)	-23.64*** (-17.74)	2.306 (0.53)	3.295 (0.77)
created_at_hour dummies?	No	Yes	No	Yes
Controllers Used	Search	Search	All	All
_cons	150.1*** (336.67)	133.2*** (116.37)	606.5*** (420.98)	491.0*** (130.17)
<i>N</i>	2345617	2345617	3858943	3858943

t statistics in parentheses

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

Set of regressions which examine the effect of insertions and deletions in the code on the number of views that a user will have on page p in the one day time span after viewing page p . The after_commit variable is equal to 1 if the current observation occurs after the commit that it corresponds to. All observations are within a four day window of the commit (2 days before and 2 days after) and contain only users who have seen page p both before the commit and after the commit.

The variables insertionspercentile and deletionspercentile are calculated as the percentile rank of the number of insertions and deletions for the current commit, respectively. The created_at_hour dummies correspond to dummies equaling 1 if the hour is equal to i and 0 otherwise for $i \in [1, 24]$.

The dataset is either confined to the search controller, or uses all the controllers in the following list: communication, customs, my_panjiva, profile, project, us_exports, and us_imports controller.

4.3 Differential Controller Effects

One might also want to examine how users change their behaviors depending on the controller for which a change is made. Perhaps users react differently to changes in different controllers. For example, the search controller is the most heavily used and is the controller that is most frequently a “return controller” (a page that a user will repeatedly visit). However, other controllers tend to be visited less often and tend not to be repeatedly visited. One would expect that changes in different types of controllers would have different affects on user activity on those controllers.

Thus, the following regression specification was used to determine the effects of each controller:

$$\begin{aligned} num_views_day_later_{it} &= \beta_0 after_commit_{it} + \bar{\beta}_1 \overline{controllers}_{it} \\ &+ \bar{\beta}_2 \overline{controllers}_{it}^T \times \overline{after_commit}_{it} \\ &+ \beta_3 hour_dummies_{it} + \epsilon_{it} \end{aligned} \quad (6)$$

Where $\overline{controllers}$ is a vector of dummies of possible controllers. For example, if the current observation corresponds to an activity on the search controller then $\overline{controllers}_{search} = 1$ while $\overline{controllers}_j = 0$ for all $j \neq search$. However, there is no coefficient in this regression that gauges the impact of controller k on the user activity after a commit. Notice that before a commit, $after_commit = 0$ and the interaction terms for the $\bar{\beta}_2$ coefficient are all zero. However, one controller dummy variable, namely the controller dummy corresponding to the k th controller is equal to 1. After the commit on controller k , the $after_commit$ variable will be equal to 1, as will the interaction term corresponding to controller k . Thus, in order to find the difference between these two effects, one needs to take $\beta_{2k} + \beta_{1k} + \beta_0 - \beta_{1k} = \beta_{2k} + \beta_0$.

Thus, to find the difference in the number of views due to a code change in controller k , one has to examine the sum $\Gamma_k = \beta_{2k} + \beta_0$. Note that for significance testing, one uses the null hypothesis $H_0 : \beta_{2k} + \beta_0 = 0$, and standard errors can be calculated with the formula:

$$SE_{sum} = \sqrt{SE_{controller_{2k}}^2 + SE_{after_commit}^2 + 2Cov(controller_{2k}, after_commit)} \quad (7)$$

These differential controller effects Γ_k are given in table 6. It is clear that each controller has a different effect. In particular, one can see why the effects for all the controllers in table 5 differ from the effects found for just the search controller in table 4. Both the My_Panjiva controllers and the Project controllers have negative and significant Γ coefficients.

Examining the differential effects, one can see that there is no significant effect to user behavior when there are changes in most of the controllers. There is a significant and negative effect for the My_Panjiva and Project controllers, and a positive effect for the Search controller. Moreover, the magnitude of the coefficients differ greatly across controllers. The magnitude of the Project controller’s Γ is much larger than the Γ for the search controller.

Therefore, it seems that there is significant variation in user responses across different controller types. Interestingly, the My_Panjiva and Project controllers do not tend to be repeatedly used, unlike the Search controller which has many repeat views. One hypothesis could be that code changes have a positive effect on user activity only on pages which users repeatedly use. However, one could also notice that controllers with more repeated views are more likely in general to have a higher number of views on any given day. Thus, the difference in the number of views after a commit could be due solely to the fact that a viewer visits a certain page more often.

Thus, to determine the effect of commits on controllers with different frequencies of use, one must include another variable that controls for how often a controller is usually viewed. A new variable, `num.controller.views` will be created for each observation. The variable will

Table 6: Differential Controller Effects on After Commit User Activity

	(1)	(2)
	No Hour Controls	With Hour Controls
Communication	-2.673 (-0.50)	5.111 (0.98)
My_Panjiva	-41.718* (-1.98)	-54.530*** (-2.68)
Profile	-8.167 (-1.30)	-6.339 (-1.04)
Project	-410.438*** (-235.55)	-388.496*** (-228.27)
Search	5.541*** (4.37)	1.516 (1.23)
US_Exports	10.318 (0.28)	-28.946 (-0.80)
US_Imports	-12.487 (-0.82)	-8.73 (-0.59)
<i>N</i>	3858943	3858943

t statistics in parentheses

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

be equal to the total number of times the current user views the current controller (minus the num_views_day_later variable so as to not introduce explicit multicollinearity).

The new regression specification will now contain the num_controller_views variable and also an interaction term between num_controller_views and after_commit to see how the number of controller views affects the success of a commit. Results are shown in table 7.

From the table, one can see that both insertions and deletions increase the effectiveness of a commit (their interaction term coefficients with after_commit are positive and statistically significant). This falls in line with previous results from the micro-level dataset. Interestingly, controllers with higher number of views tend to respond poorly to commits, ceteris paribus.

The coefficient on the num_controller_views * after_commit interaction term provides the change in slope after the commit. Past regression specifications did not account for the fact that already popular pages would have more views after a commit simply because of its past popularity. This specification, however, examines how commits change the number of views depending on the previous popularity of a page.

The negative and significant interaction term coefficient implies that in fact, controlling for previous popularity, controllers which already have a high number of views tend to be adversely affected by commits. This might occur because users require adjustment time, especially for pages that they use frequently. Another explanation is that changes to infrequently used pages tend to increase user behavior more than changes to frequently used pages since users have less familiarity with the workflows of the former.

One must take these results with caution however, since multicollinearity could be lurking.

Table 7: Controller Popularity Effects

	(1)	(2)
	num_views_day_later	num_views_day_later
after_commit	-13.79*** (-9.02)	-9.618*** (-6.40)
insertionspercentile	-108.8*** (-52.39)	-104.9*** (-51.40)
deletionspercentile	-5.528** (-2.70)	-6.231** (-3.10)
insertionspercentile *	63.96***	54.02***
after_commit	(20.91)	(17.96)
deletionspercentile *	67.98***	67.61***
after_commit	(22.53)	(22.79)
num_controller_views	0.0475*** (1780.26)	0.0469*** (1751.65)
num_controller_views *	-0.0138***	-0.0134***
after_commit	(-333.01)	(-328.18)
Hour Dummies?	No	Yes
_cons	-47.43*** (-44.85)	-69.41*** (-26.29)
<i>N</i>	3858943	3858943

t statistics in parentheses

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

Its possible that the number of controller views is correlated with the number of views the day after an observation, even though explicit dependency has been removed. Thus, further research needs to be done in order to support this hypothesis fully.

5 Conclusion

This paper examined the effect of code changes on user activity. Total user activity tended to increase when the total number of code insertions increased. However, both total and average user activity tended to decrease when more files were changed. This phenomenon most likely occurs because of the high volume of code changes on Panjiva’s website. Thus, it seems likely that users respond positively to higher quality code, or alternatively, that users respond negatively to low quality code.

Exploiting the granularity of Panjiva’s dataset, the paper found that code changes on particular website pages, such as the Search page, resulted in a higher number of repeat views. The micro-level dataset also reinforced the hypothesis that user activity tends to increase with an increase in the number of lines of code, *ceteris paribus*. Finally, the paper found that code changes tended to have negative affects on popular pages, possibly because of the time required to adjust to these changes.

Code changes seem to have an ambiguous effect on user activity in general. They can be

successful only with a confluence of factors. In particular, high quality changes tend to have a better chance of positively influencing users. Also, changes to highly trafficked pages can have negative effects on user behavior. Thus, it seems that change by itself does not tend to be a good strategy in general. Changes need to be targetted and of high quality in order for them to be successful.

A Appendix: Explanation of Dataset

The original event and activity log datasets are tables in a PostgreSQL database. Panjiva uses a Ruby on Rails framework for its entire website, and stores ruby objects in the PostgreSQL database. The activity and event logs each contain certain attributes that can be queried for. The attributes are listed below:

Table 8: Activity Log Attributes

Attribute Name	Attribute Type	Comment
<i>id</i>	integer	Unique id for each record
<i>user_account_id</i>	integer	Id of the associated user account
<i>controller</i>	string(255)	Controller viewed
<i>action</i>	string(255)	Specific action of the controller viewed
<i>model_id</i>	integer	Id of the database model
<i>status</i>	string(255)	Status of completion of request
<i>created_at</i>	datetime	Time the record was created
<i>ip_address</i>	string	
<i>session_id</i>	integer	Id of the browser session

The event log attributes are very similar except there is less informative data. In particular, the event logs do not have data on the controller or actions viewed by a given user. Note that activity logs only record actions performed by users who have registered with Panjiva and have signed into the website. Event logs tracks actions performed both both registered users and by those who have not logged in. Consequently, the event log is much larger, but has less informative data.

The event logs contain about 124 million records, while the action logs contain 13 million records. The data in the PostgreSQL database was parsed into macro level and micro level datasets.

A.1 Macro-Level Dataset

The macro-level dataset contains aggregated count statistics for each active day after 7/14/2011. The dataset was created by making a SQL query to find the total count of actions on the event and activity logs, as well as the total distinct users. This populated the *activitylog_count*, *activitylog_distinct_user_count*, *eventlog_count*, and *eventlog_distinct_user_count* variables (since these were broken down by day, the *date* and *ay_of_week* variables were also created).

To get information about commits, such as the insertions, deletions, and files changed for a given day, the version control system used by Panjiva was mined. Panjiva uses git and Github to store and track changes in its source code. In order to find the total number of lines and files changed in a given day, the master git repository (the version of the code which is public and launched to production) was programmatically scraped for information. Note that the variable *lineschanged* counts every line that was changed, whether an insertion or deletion,

and is simply equal to *insertions* + *deletions*. The variable *fileschanged* is the number of distinct files for which at least one line was inserted or deleted.

Attribute Name	Attribute Type	Comment
<i>date</i>	<i>date</i>	Date of the record
<i>day_of_week</i>	<i>integer</i>	Sunday = 0 and Saturday = 6
<i>lineschanged</i>	<i>integer</i>	Total number of lines changed
<i>fileschanged</i>	<i>integer</i>	Total number of files chanegd
<i>insertions</i>	<i>integer</i>	Number of lines changed that were insertions
<i>deletions</i>	<i>integer</i>	Number of lines changed that were deletions
<i>fileschanged_percentile</i>	<i>float</i>	See description below
<i>insertions_percentile</i>	<i>float</i>	See description below
<i>deletions_percentile</i>	<i>float</i>	See description below
<i>activitylog_count</i>	<i>integer</i>	Count of all activity log actions in the current day
<i>eventlog_count</i>	<i>integer</i>	Count of all event log actions in the current day
<i>activitylog_distinct_user_count</i>	<i>integer</i>	Number of distinct users that appeared in the activity log
<i>eventlog_distinct_user_count</i>	<i>integer</i>	Number of distinct users that appeared in the event logs

The percentile variables were created by sorting the base variable and dividing the index by the number of days. For example, if the number of lines changed was [10, 5, 2, 3, 12] for days 0 through 4 respectively, then the list would first be sorted to [2, 3, 5, 10, 12] and the day corresponding to two lines changed would have a percentile of $0/5 = 0$, while the day corresponding to three would have a percentile of $1/5 = 0.2$. This would continue upwards until the last day with a percentile of $4/5 = 0.8$. Thus, the percentiles for days 0 through 4 would be [3/5, 2/5, 0/5, 1/5, 4/5].

The percentile variables are simply the ranked percentile of a given observation in comparison to the entire distribution. In essence, one can think of the percentile transformation as mapping an arbitrary distribution to a uniform distribution. The transformation was done for *fileschanged*, *deletions*, and *insertions*.

A.2 Micro-Level Dataset

The micro-level dataset was obtained in a similar manner, however, much more attention was paid to each commit. First, the git repository was searched for all commits in the period from 7/14/2011 until 11/26/2012 concerning a given page of the site. Each of these commits had a timestamp, the exact lines of code changed, and the author of the change. For each commit, the timestamp was examined, and the activity logs were searched for all actions that occurred within two days of the commit's timestamp (a four day window corresponding to two days before and two days after) and occurred on a page changed by the commit. For each of these observations, the number of total views of the page in the twenty four hours following the observation's date were recorded in *num_views_day_later*. Dummies for the controller were created for each observation.