Investigating the Relationship Between Testing and Pull Requests

Matthew Chung, Henry Truong, Hiu Fung Kevin Chang

***Abstract*— Testing code is a common practice used by many developers across various projects. Pull requests are a method used to merge developer code into the master code. This paper explores whether projects that enforce testing have a higher chance of code being merged into production by observing if overall test density is maintained or improved through an analysis of the TravisTorrent data set and its pull requests on GitHub.**

I. INTRODUCTION

The research question being investigated in this paper is whether or not a relationship exists between testing and pull request merges. This research, which can be found at: `https://github.com/cmput402-w19/project-Team8`, aims to improve developer work-flow by understanding how they can improve the chances of their code being merged successfully. Testing code is one of many important steps to reinforce the quality of a piece of software. Not only does testing code ensure that a program is working as expected, it also ensures that the code being written is in fact, testable. We can see from this, that there are multiple positive reasons to test code. For code to make its way to release, it must be merged in first using a process called pull requests. Pull requests allow code owners to review the changes that a developer wants to merge, this process gives code owners an opportunity to request changes, review, or reject this code before it can make its way into the master branch. TravisTorrent provides us with a large database of commits from GitHub and their properties which was acquired through GHTorrent and Travis CI build logs. We aim to answer this question by selecting repositories and analyzing different attributes regarding the test density of their pull requests from this TravisTorrent data set.

II. HYPOTHESIS

Since testing is an essential step in production level development, test density should not decrease in general and developers should always add tests for the corresponding feature being implemented. Our hypothesis states that, maintaining or increasing a project's test density positively correlates with pull request merges, and conversely, decreases in overall testing correlates with pull request rejections.

III. RELATED WORKS AND TOPICS

The following section discusses other works that have been conducted which are related to our research topic. These papers and previous works have been selected due to their use of similar data sets and for having research questions that explore similar goals. Topics the authors cover includes pull request reviews, project requirements, continuous integration and standards which different companies enforce. While these papers explore the relationship between continuous integration and developer productivity and processes, there is no specific discussion on whether or not testing has a positive correlation with merged pull requests. Testing is acknowledged as part of the work-flow in continuous integration and previous work has demonstrated that a broken build due to test failures is less likely to have pull requests merged in. However, there is no specific statistical analysis and research that overlaps entirely with ours.

A. Continuous Integration and Quality Assurance: a case study of two open source projects

Holk and Jrgensen discuss continuous integration practices of FreeBSD and Mozilla projects as a case study. An example of practices employed by Mozilla related to testing is that simple tests must be run before a developer commits their work [1]. There is no specific policy these projects employ for mandatory testing. Contributors do not need tests for features developed to prove correctness. However, the main expectation is that builds must never break when work is committed by developers. The insights provided by Holk and Jrgensen relates to our work since we wish to find certain policies relating to testing for different project repositories. Another area of interest in this paper is the work-flow developers follow to merge feature code into production, this helps us understand whether or not testing plays a role in production code merges.

B. Oops, My Tests Broke the Build:An Explorative Analysis of Travis CI with GitHub

Beller, Gousios and Zaidman's work further investigates continuous integration due to it being a strong practice for software development. The previous work's research is guided by the effects of testing and Travis CI. Beller, Gousios and Zaidman's work was selected due to several of their research questions relation to the topic using a different context. Their work also provides further insights on policies regarding testing and code merges. An example of a related research question found in this paper is, "Are tests a decisive part of CI?" [2]. The results concluded from their investigation are that testing is a core stage of continuous integration, something which further guides the motivation for our research question. Due to the importance of the testing stage of continuous integration, it is also one of the main reasons for build failures [2]. For similar metrics to our

research, number of tests was found to be highly dependent on languages that are dynamically typed. These were found to have more tests than statically typed languages, however, there was no evidence to support an increase in merges to production code [2]. For the research question relating to this paper, it is found that test failures are generally ignored because they do not result in a failing build.

C. TravisTorrent: Synthesizing Travis CI and GitHub for Full-Stack Research on Continuous Integration

Beller, Gousios and Zaidma look into the effects of continuous integration on software quality and productivity affected by continuous integration. Research questions relating to our work includes, "What are CI best practices that successful projects employ?" and "Does a broken build really negatively affect other developers productivity, as is often claimed?" [3]. These questions drove Beller, Gousios and Zaidma to build the TravisTorrent data set to better help researchers do analysis on builds of repositories [3]. Their work and motivations for developing the TravisTorrent data set opens up many possibilities for researching build information. Our research questions and motivations are inspired from their motivations, however, we want to streamline our focus towards pull requests. One goal that differentiates our research question and motivations from the creators of TravisTorrent is that rather than attempting to quantify tests and build successions effects on continuous integration, we want to discover how it directly affects pull request evaluation. We also want to better understand how these results can improve developer work-flow of production code merges.

D. Quality and Productivity Outcomes Relating to Continuous Integration in GitHub

This work investigates continuous integration and aims to answer the questions of whether software quality in teams is affected by continuous integration. These questions align with our goals of potentially improving work-flow of developers as well as increasing the chances of merging pull requests when testing is performed. The results from this investigation concludes that there is a positive effect found from using continuous integration on the number of merged pull requests [4]. Extending this, continuous integration was found to also have a significant negative effect on the number of rejected pull requests. This paper uses continuous integration in place of our test density and finds conclusive evidence that supports an improvement to the success of developer pull requests, which we also aim to prove for testing as a whole.

E. Wait For It: Determinants of Pull Request Evaluation Latency on GitHub

Filkov and the authors of this paper investigate pull request evaluation, specifically pull request latency which represents the time interval between a pull request creation and close date. Pull-based development has become a widely adopted practice to merge incoming code changes into production ready code and continuous integration is utilized to assist in this evaluation [5]. Filkov investigates factors that contribute to pull request latency. In their discussion of the evaluation process of pull requests, automatic tests are run in continuous integration and the results determine if further review or rejection is required. Results from this study found that including test cases and active areas of the project result in higher pull request latency [5]. While this research question differs from our work, it further shows the importance of tests and demonstrates the effects they have on the pull request evaluation process, this is similar to our research which explores whether test density has a relationship with pull requests.

IV. METHOD

With the TravisTorrent data set, our goal was for 50% of the data to be merged pull requests and the other 50% of the data to be non-merged pull requests. Initially, we sampled 30 random repositories, each with 30 pull requests within the data set. We noticed the majority of the pull requests within our sample were merged while few were not merged which made our sample set disproportionately unbalanced. To achieve the balanced proportion of 50% merged and non-merged pull requests, we did the following:

A. Repository Selection

1) We queried the database for repositories with the most rows containing 'unknown', 'fixes_in_commit', or ' ' (blank) in the 'git_merged_with' column.
2) We then checked each pull request to confirm the merge status using the GitHub API since the pull requests above may not necessarily be rejected.
3) We chose the 30 repositories with the highest amount of non-merged pull requests.
4) Finally, we looked at the repositories and categorize them as: 1. Repository did not mention how to contribute for pull requests 2. Repository has a contribution guide and does not require tests 3. Repository has a contribution guide and required tests for future review. Table I shows the repositories we selected and the corresponding policy category mentioned above.

B. Data Creation

For each repository selected, the first step was to go through every failed pull request commit. Since rows containing 'unknown', 'fixes_in_commit', or ' ' (blank) in the 'git_merged_with' column do not imply that the pull request was merged, we had to query the GitHub API to ensure this pull request was in fact not merged. We then used the GitHub API to find the common ancestor commit and with these two commits, we were able to query the TravisTorrent data set and calculate test density differences. The illustration on the left of Figure 1 shows the commits we are comparing when the pull request was merged.

Since most pull requests we found are merged, we had to balance the data set to include non-merged pull requests. To accomplish this, we considered pull requests with changes requested or reviews to be not merged. The logic behind

	Repository Name	Contribution Type
1	heroku/heroku	1
2	MagLev/maglev	1
3	Shopify/liquid	1
4	opal/opal	1
5	travis-ci/travis-core	1
6	HubSpot/Singularity	1
7	datastax/java-driver	1
8	heroku/heroku-buildpack-ruby	1
9	Shopify/identity_cache	1
10	square/picasso	2
11	zendesk/samson	2
12	rspec/rspec-mocks	2
13	celluloid/celluloid	2
14	rspec/rspec-core	2
15	rackerlabs/blueflood	2
16	owncloud/android	2
17	test-kitchen/test-kitchen	2
18	activeadmin/activeadmin	2
19	sanemat/tachikoma	2
20	querydsl/querydsl	2
21	xetorthio/jedis	3
22	jnicklas/capybara	3
23	adhearsion/adhearsion	3
24	expertiza/expertiza	3
25	prawnpdf/prawn	3
26	hw-cookbooks/graphite	3
27	projectblacklight/blacklight	3
28	geoserver/geoserver	3
29	chef/omnibus	3
30	thoughtbot/shoulda-matchers	3

TABLE I

CONTRIBUTION TYPES: 1. THE REPOSITORY DID NOT MENTION HOW TO CONTRIBUTE TO THE REPOSITORY 2. THE REPOSITORY HAS A CONTRIBUTION GUIDE AND DOES NOT REQUIRE TESTS 3. THE REPOSITORY HAS A CONTRIBUTION GUIDE AND REQUIRED TESTS

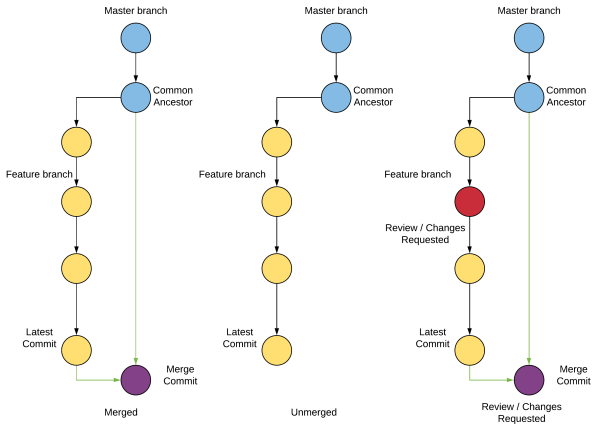

Fig. 1. Left: The pull request is merged. Middle: The pull request is not merged. Right: The pull request is merged but has a review or changes are requested.

this was that if a pull request has no issues, then there will not be any reviews and/or changes requested. A possible review however, would be a request to add or improve that newly implemented feature's test cases. If a pull request had a review or change request, we compared the test density between the latest commit and the test density of the commit before the review was made. The illustration on the right of Figure 1 shows the commits we are comparing when there was a review or changes requested.

We calculated the differences of test density and number of tests ran by querying the TravisTorrent database for these columns on the two different commits we would be comparing:

- gh_test_lines_per_kloc: Number of lines of test cases added per thousand lines of code.
- gh_test_cases_per_kloc: Number of test cases added per thousand lines of code.
- gh_asserts_cases_per_kloc: Number of assertions added per thousand lines of code.
- tr_log_num_tests_run: Total number of tests ran.
- tr_log_num_tests_ok: Number of tests passed.
- tr_log_num_tests_failed: Number of tests failed.

The process of data generation is also detailed in Algorithm 1 and Table II shows the column names in our result file for each repository.

	Column Name
1	RepoName
2	PrNumber
3	Before Line Density
4	After Line Density
5	Line Density Difference
6	Before Test Case Density
7	After Test Case Density
8	Test Case Density Difference
9	Assert Test Cases Before
10	Assert Test Cases After
11	Assert Test Cases Difference
12	Before Num Tests Run
13	After Num Tests Run
14	Num Tests Run Difference
15	Before Num Tests Pass
16	After Num Tests Pass
17	Num Tests Pass Difference
18	Before Num Tests Failed
19	After Num Tests Failed
20	Num Tests Failed Difference
21	PR Merged With

TABLE II

COLUMNS OF EACH RESULT TABLE WE GENERATE.

C. Generating Graphs

The data generated was saved as a separate CSV file as illustrated in Table III for each GitHub repository. We used the Python library Pandas to read each CSV file into a data frame. The data frames were then concatenated into a larger data frame containing results of all the repositories. Python Matplotlib was used to generate the scatter plot.

	RepoName	PrNumber	Line Density Difference	Test Case Density Difference	Assert Test Cases Difference	PR Merged With
1	HubSpot/Singularity	975	-0.05690785545699839	-0.001948724683280112	-0.009578045502202315	Merged
2	HubSpot/Singularity	993	-2.665581350804004	-0.06932479175510053	-0.4182547871645994	Merged
...	...	...	...	...	...	...
29	HubSpot/Singularity	1071	-0.023849110299991594	-0.0008320666480594596	-0.004030322826501731	Review
30	HubSpot/Singularity	1073	0.0	0.0	0.0	Review

TABLE III

EXAMPLE RESULT TABLE OF HUBSPOT/SINGULARITY

Algorithm 1 DataGeneration

Input: $R \in Repository_1 \dots Repository_{30}$
Output: $result\_file_1 \dots result\_file_{30}$

$failed\_results \leftarrow []$
$success\_results \leftarrow []$
for $repository\ r \in R$ **do**
  $i \leftarrow 1$
  for all $failed\ pull\ request\ f$ **do**
    $non-merged \leftarrow$ is_merged(f)
    if $non-merged$ **then**
      $latest, parent \leftarrow$ get_pull_request_info(f)
      $result \leftarrow$ test_density$(latest, parent)$
      $failed\_results.append(result)$
      if $len(failed\_results) <= 15$ **then**
        break
      end if
    end if
  end for
  for all $merged\ pull\ request\ m$ **do**
    $have\_review \leftarrow$ get_review(m)
    $changes \leftarrow$ get_changes(m)
    if $have\_review$ **or** $changes$ **then**
      $latest, parent \leftarrow$ get_pull_request_info(m)
      $before\_review \leftarrow$ get_review_commit(m)
      $result \leftarrow$ test_density$(latest, before_review)$
      if $len(failed\_results) <= 15$ **then**
        $failed\_results.append(result)$
      end if
    else
      $latest, parent \leftarrow$ get_pull_request_info(m)
      $result \leftarrow$ test_density$(latest, parent)$
      $failed\_results.append(result)$
      if $len(success\_results) <= 15$ **then**
        $success\_results.append(result)$
      end if
    end if
  end for
  $result \leftarrow success\_results \cup failed\_results$
  write $result$ to $result\_file_i$
  $i += 1$
end for

V. RESULTS

After the result files were generated, we noticed that the columns 'tr_log_num_tests_run', 'tr_log_num_tests_ok', and 'tr_log_num_tests_failed' in our results were mostly empty due to the data being unavailable within the TravisTorrent data set. This gave us very few rows of usable data therefore we could not use them as attributes in our correlation analysis.

Using the aforementioned methods we have analyzed 861 unique pull requests from 30 different repositories. Utilizing the Python libraries Matplotlib and Pandas, we plotted the data mined from these pull requests, with the difference in test case density before and after the pull request on the x-axis and assert case density difference on the y-axis. Figure 2 shows the complete scatter plot of these pull requests, while Figure 3 has a focused view of the data after cropping any profound outliers for improved granularity. The results which are demonstrated in this data are very cluttered with a sizable cluster of data points around zero for both merged and non-merged pull requests. Further analysis revealed that approximately 70% of merged pull requests increased or maintained both test case density and assert case density. The opposing analysis showed that approximately 29% of non-merged pull requests decreased both assert case density and test case density. Unfortunately, 23% of merged pull requests also decreased assert case density and test case density, for this reason we elected to run a correlation test to determine significance.

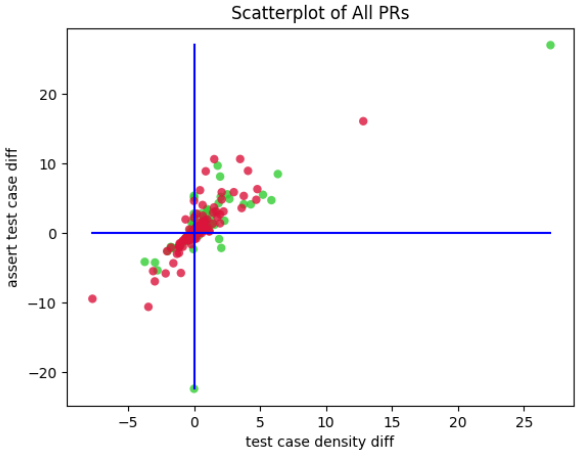

Fig. 2. Overall scatter plot of 861 pull requests

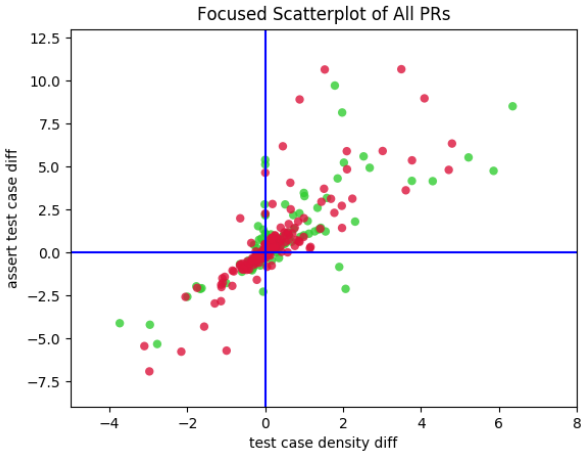

Fig. 3. Focused scatter plot excluding outliers

We used Numpy to perform Point-Biserial correlation analysis to compare the merge status with test case density difference and assert case density difference respectively. The reason we used Point-Biserial analysis is due to the comparison being made between a dichotomous categorical variable, merge status, and a continuous variable represented by the density differences. The correlation coefficient for test case density difference was calculated to be 0.030713938637360074, using the null hypothesis which states that there is no relationship between merged pull requests and test case density along with our calculated p-value of 0.368047835420104, we are unable to reject this null hypothesis. The second null hypothesis states that there is no relationship between merged pull requests and assert case density, our analysis revealed a similar r-value of 0.009116331437595618 and p-value of 0.7893801260274774 which means we also cannot reject this null hypothesis.

We performed further analysis, dividing the repositories analyzed by investigating the merge policies recommended or enforced in each project, this was done so that we could see if there was any noticeable difference in the resulting data. We categorized these repositories into two categories, ones which had merge policies that required testing and those without this policy. Using our graphing script, we plotted the results into a scatter plot so that we could observe if there were any obvious variations when compared to our original graph. The graph of repositories with testing policies can be seen in Figure 4. We then calculated to see how many of these merged pull requests with merge policies increased test case and assert density; the resulting values showed that 73% of merged pull requests increased test case and assert density while 29% of non-merged pull requests decreased both test case and assert density. This shows a marginal improvement towards our original hypothesis compared to the overall data set, however, the change is relatively minor and not as large of a jump as we originally expected. The statistics for pull requests without testing contribution policies found in Figure 5 shows that 67% of merged pull requests increase test case and assert density, while 28% of non-merged pull requests decrease test case and assert density, showing that both numbers go down when there is no policy by a small percentage.

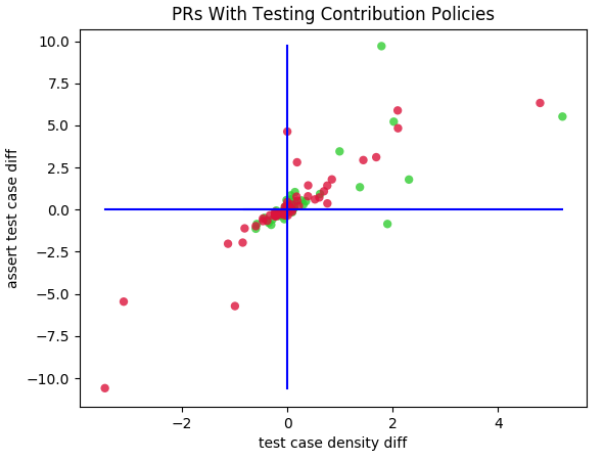

Fig. 4. Scatter plot pull requests with contribution policies on testing

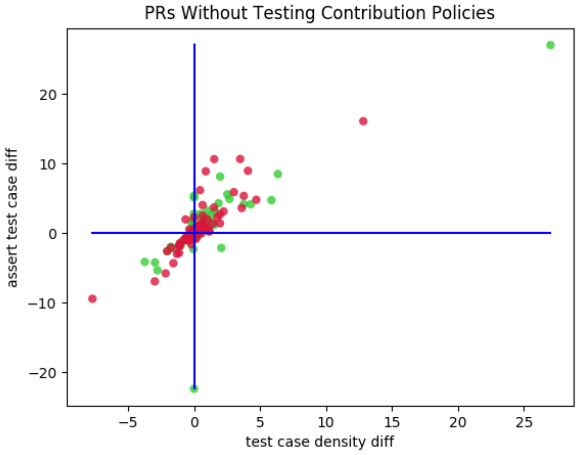

Fig. 5. Scatter plot pull requests without contribution policies on testing

VI. THREATS TO VALIDITY

Mentioned in the Method section was the issue found with the various merge statuses in TravisTorrent, the columns had various possible values such as 'unknown' and even blank ' ' or empty values. When we investigated these particular pull requests further we found that there was complete inconsistency with the pull request outcome, several of these pull requests were rejected, while others were in fact, merged to master. Due to this poor reliability we queried the GitHub API on each pull request to establish the merge status for one row, without this extra step, our data would be entirely unreliable. Since we can see from this that the TravisTorrent data had certain questionable values, it provokes the idea

that other sections of the TravisTorrent data set may have inaccuracies. Since we do not perform the test density calculations ourselves and are thus relying on the data in TravisTorrent to be correct, this poses another threat to validity in our research, if these testing densities are not always calculated correctly, it may skew our results from the true reality.

VII. CONCLUSION

Our original goal with this research was to determine if improvements to overall testing in a project's pull requests had a relationship with successful merges. The reason this research is important is because it seeks to improve one of several obstacles that developers face throughout the development process. Merging one's code is a vital step in the overall software engineering cycle and finding any way to improve this could possibly help a substantial amount of engineers.

We investigated this by analyzing various repositories that enforced policies that were related to testing and repositories who had no testing policies in place. By looking at both merged and non-merged pull requests we gathered information that would show us if there was any relationship with the actual testing performed in those pull requests and their merge outcome. The results of our analyses showed that there was not a significant relationship between these two variables, however we believe that there is still more possibilities to be investigated on this subject.

VIII. FUTURE WORK

Replicating this study using a larger sample size would likely yield similar results, however, combining this with an analysis of code coverage tools i.e. Cobertura over the pull request history of many repositories is something which may present interesting results. This type of analysis would be interesting since it looks at the lines of source code executed before and after pull requests, a metric which if significant, would be equally valuable to improving developer work-flow.

REFERENCES

[1] J. Holck and N. Jergensen, "Continuous Integration and Quality Assurance: a case study of two open source projects," Australasian Journal of Information Systems , vol. 11 no. 1, Nov 1, 2003. [online]. Available: https://journal.acs.org.au/index.php/ajis/article/view/145.[Accessed Feb 15, 2019].

[2] M. Beller, G. Gousios and A. Zaidman, "Oops, My Tests Broke the Build: An Explorative Analysis of Travis CI with GitHub," in 2017 IEEE/ACM 14th International Conference on Mining Software Repositories, MSR 2017, Bueno Aires, Argentina. 20-21 May, 2017 [online]. Available: IEEE Xplore, https://ieeexplore.ieee.org/document/7962385.[Accessed Feb 15, 2019].

[3] M. Beller, G. Gousios and A. Zaidman, "TravisTorrent: Synthesizing Travis CI and GitHub for Full-Stack Research on Continuous Integration," in 2017 IEEE/ACM 14th International Conference on Mining Software Repositories, MSR 2017, Bueno Aires, Argentina. 20-21 May, 2017 [online]. Available: IEEE Xplore, https://ieeexplore.ieee.org/document/7962393.[Accessed Feb 15, 2019].

[4] B. Vasilescu, Y. Yu, H.Wang, P. Devanbu and V.r Filkov, "Quality and Productivity Outcomes Relating to Continuous Integration in GitHub," in Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, Bergamo, Italy. 30 Aug - 04 Sept, 2015 [online]. Available: dl.acm.org https://dl.acm.org/citation.cfm?doid=2786805.2786850. [Accessed Feb 15, 2019].

[5] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu and V. Filkov, "Wait For It: Determinants of Pull Request Evaluation Latency on GitHub," in 2015 12th Working Conference on Mining Software Repositories, Florence, Italy. 16-17 May, 2015 [online]. Available: IEEE Xplore, https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7180096. [Accessed Apr 13, 2019].

[6] M. Beller, G. Gousios, A. Zaidman. (2017) TravisTorrent: Synthesizing Travis CI and GitHub for Full-Stack Research on Continuous Integration