Continuous Integration Build Breakage Rationale: Travis Data Case Study

Panagiotis Dimitropoulos[1], Zeyar Aung[2], Davor Svetinovic[3]

[1,2,3]Khalifa University of Science and Technology, Masdar Institute, Abu Dhabi, United Arab Emirates
[1]{pdimitropoulos, [2]zaung, [3]dsvetinovic}@masdar.ac.ae

Abstract:Continuous Integration (CI) has a prominent rolein software engineering. However, little research that involvesquantitative results has been done upon the verifiable outcomesof this practice. TravisTorrent, a freely available data set basedon Travis CI and GitHub provides deep analysis of the projectsource code, process and dependency status of 1,359 projects thatuse CI. In this paper, we analyze this data set and explore thefeatures in order to get the information about the factors thataffect build breakage.

Keywords: continuous integration, Travis CI, GitHub, TravisTorrent

I. INTRODUCTION

Modern software has led to the necessity for the collaborationbetween tens and hundreds of developers in order todevelop the software systems of ever increasing size in adistributed fashion. In the Open Source Software (OSS) development,teams are globally distributed, and they are not evenunder a centralized management. The only way to preserve themarket necessities in an agile and organized way, with limitedcentralized control, is to perform a variety of technologicalapproaches, including enabling process automation. AlthoughOsterweil was the pioneer of the idea of process automationlong time ago [9], the increased demands of later trends suchas OSS, distributed development or cloud computing, havedriven numerous innovations on this area. Git repositories,forking, pull requests and Continuous Integration (CI) aresome of the examples of such innovations in the area ofdistributed collaborative technologies. Nonetheless, because ofthe rapid changes, it is really hard to derive results about theeffects on product quality outcomes. External factors such ascode size, contributors diversity and user interest can shapeoutcomes and thus, exporting the effects of process innovationcan be a challenge.

GitHub is one of the most popular web-based Git repositoryhosting service. It offers all of the distributed version controland source code management (SCM) functionality of Git aswell as adding its own features. It provides access controland several collaboration features such as bug tracking, featurerequests, task management, and wikis for every project. Followingthe pull-based development model [1], GitHub allowsany developer to contribute to any project in the form ofpull requests. A pull request is a suggested code change thatmost of the time reflects a previously submitted modificationrequest or issue. Core developers of a project can reviewthese pull requests and accept them if they believe that theyprovide relevant contributions. However, projects that are morepopular, automatically attract more contributors and thus theyreceive more pull requests. Before merging a pull requestinto the main development branch, the projects' integrators(core developers) have to build, test and review the proposedcontribution. This can slow down the development progressof the project since the integrators will not be able to resolveall the pull requests efficiently. This is where process automationcan provide value. Therefore, CI, one of the distributedcollaboration innovations, is being used to automatically buildand deploy the software in a virtual environment, often calleda sandbox, and to automatically run a set of tests. Thisautomation process is meant to increase both productivity(more pull requests accepted) and quality (the accepted pullrequests are already automatically checked).

CI builds can be either positive (passed) or negative (fail,errored, canceled) and the information that they provide has avery important role in the overall development progress. In thisstudy, we aim to reveal the correlation between the breakage ofthe CI builds and numerous factors that could possibly affectthe build outcome (e.g., team size, programming language,source code churn, test code churn, test density, number ofcomments, test duration). In particular, we explore if theconsiderable changes in a project's churn and project's filesaffect the build status, and if the test time, build time andbuild setup time are a considerable variable that can producemore broken builds.

The purpose of this study is to better understand thefactors behind build brakeage in various pull-based softwaredevelopment efforts. This will help developers and managersto learn from those past failures and to device better technicaland/or management strategies in order to avoid such failuresin their own development endeavors.

II. RESEARCH METHOD

A. Data Collection and Preprocessing

We use the TravisTorrent data set [4], a freely available dataset based on Travis CI and GitHub, which provides easy

978-1-5386-0514-1/17/$31.00 ©2017 IEEE

accessto over 1000 projects. Unique to TravisTorrent is that each ofits 2,640,825 Travis builds is synthesized with meta data fromTravis CI's API, the results of analyzing its textual build log,a link to the GitHub commit which triggered the build, anddynamically aggregated project data from the time of commit extracted through GHTorrent [7]. The general data structure is as follows: there are 55 data fields and each data point (row) represents a build job executed on Travis while incorporating information from three different resources. The project' git repository (prefixed git_), data extracted from GitHub through GHTorrent (prefixed gh_), and data from Travis's API and an analysis of the build log (prefixed tr_). We expect that the data set provided is peril free, and it does not suffers from threats such as: possible issues with data gathering, no validation, and unrefined models [5]. In order to efficiently find the reasons of why builds break, we applied modifications in the given data set as presented in Figure 1.

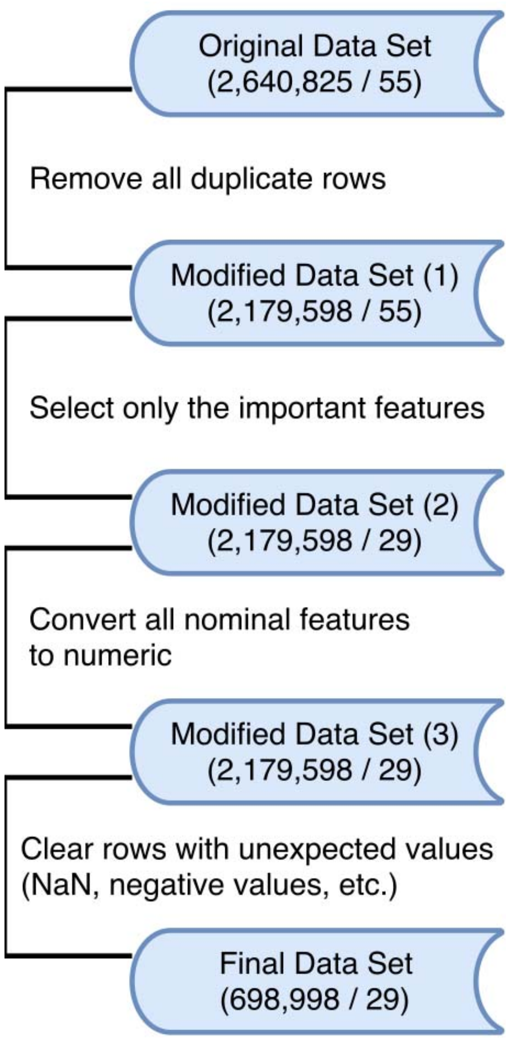

Fig. 1. Data preprocessing flowchart. The values inside the parenthesis represent the volume of the data in rows and columns.

1) While investigating the data set we identified that many of the rows were identical. Thus, we erased all the duplicates.

2) From the 55 data fields, we selected only those who we thought that were relevant and could possibly have some association with the build status (28 data fields plus the build status). Those fields are further explained later on.

3) We converted all the nominal features to numeric in order to be able to efficiently apply data mining algorithms.

4) We removed all the rows which the builds i) had no tests executions (this behavior comes in contrast with the development practice of CI [3]), ii) had values of specific fields set as NaN (not a number), iii) had values of specific fields that should be positive (e.g., build duration) set as negative. The critical amount of data reduction on this step unwittingly confirms the findings of Beller et al. [2], which although it is specified in IDEs, it points out that the majority of projects and users do not practice testing actively.

In order to better understand the pattern of our data we tried to visualize them into 2 dimensions. We performed Principal Component Analysis (PCA), a technique used to emphasize variation and bring out strong patterns in a data set. It's often used to make data easy to explore and visualize. Our output is shown in Figure 2. As it can be inferred, the data seems to have no strong correlations between passed and broken builds. There are some small clusters with increased passed builds, however this is the opposite of what we look for and it will not assist us in finding the reasoning behind the broken builds.

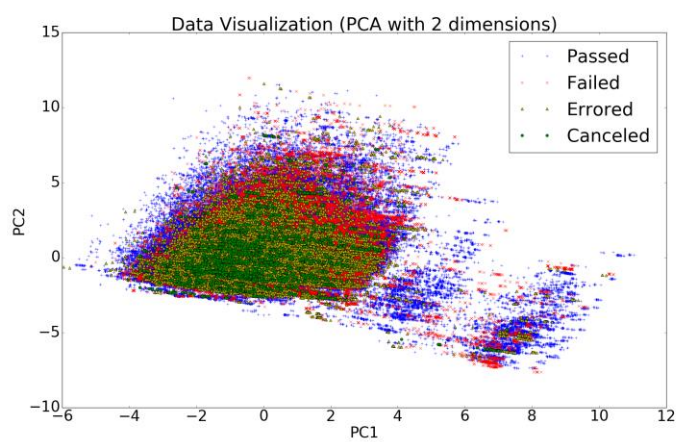

Fig. 2. Principal Component Analysis (PCA) data visualization with 2 dimensions.

B. Measures

1) Outcome: The outcome measure is the Travis CI buildstatus. We consider the build successful if the status is passedand unsuccessful if the status is failed, errored or canceled.

2) Predictors: We compute all the related measures, asdiscussed in our research questions.

git branch: Branch which the commit was committed on.

git num committers: Number of people who committed tothis project.

gh is pr: Whether this build was triggered as part of a pullrequest on GitHub.

gh lang: Dominant repository language, according to GitHub.

gh team size: Number of developers that committed directlyor merged PRs from the moment the build was triggered and3 months back.

gh num issue comments: If the commit is linked to a PRon GitHub, the number of discussion comments on that PR.

gh src churn: How much (lines) production code changed inthe commits built by this build.

gh test churn: How much (lines) test code changed in thecommits built by this build.

gh files added: Number of files added by the commits builtby this build.

gh files deleted: Number of files deleted by the commits builtby this build.

gh files modified: Number of files modified by the commitsbuilt by this build.

gh tests added: Lines of testing code added by the commitsbuilt by this build.

gh tests deleted: Lines of testing code deleted by the commitsbuilt by this build.

gh src files: Number of src files changed by the commits thatwhere built.

gh doc files: Number of documentation files changed by thecommits that where built.

gh other files: Number of files which are neither productioncode nor documentation that changed by the commits thatwhere built.

gh sloc: Number of executable production source lines ofcode, in the entire repository.

gh test lines per kloc: Test density. Number of lines in testcases per 1000 gh sloc.

gh test cases per kloc: Test density. Number of test casesin test cases per 1000 gh sloc.

gh asserts cases per kloc: Assert density. Number of assertionsper 1000 gh sloc.

gh by core team member: Whether this commit was authoredby a core team member.

tr duration: Overall duration of the build.

tr testduration: Time it took to run the tests.

tr setup time: Setup time for the Travis build to start.

tr tests ok: Number of tests passed.

tr tests fail: Number of tests failed.

tr tests run: Number of tests were run as part of this build.

tr tests skipped: Number of tests were skipped or ignoredin the build.

C. Analysis

We use a novel approach that combines two different methodsin order to get a broader understanding and to secure ourfindings. Firstly, we use k-means++ clustering method in orderto partition our 28 predictors into 4 clusters (passed, failed,errored, canceled) and get a general view of the depending featuresthat affect the build status. Secondly, we use the *Logistic Regression* to model the status of the builds. Each model isrepresentative to one of our research questions. This will helpus find which of variables are culpable for the build breakageand to what degree. In order to do that, we remove the extremeoutliers of our data by selecting only the values of each featurethat verify the expression *abs(x – x.mean) <= 10 * x:std*,where x is the values of the feature x, x.mean is the medianof the values of feature x, and x.std is the standard deviationof the values of feature x. All numeric variables were firstlog transformed (plus 0.5 to the ones that can contain zeros)to stabilize variance and reduce heteroscedasticity [6], thenstandardized (mean 0, standard deviation 1). We evaluate onthe training set in order to check the accuracy of our clusteringmodels. For the logistic regression models we perform a 10-fold stratified cross-validation to ensure that the accuracy ofour models is stable. The results of the k-means++ clusteringmodel is shown in Table I and the logistic regression modelsare shown in Table II.

III. RESULTS

In this section, we present the results to our researchquestions and we have a discussion upon them.

A. General Analysis

Our k-means clustering model (Table I) created with WEKAdata mining software [8] has unfortunately a relatively lowaccuracy. The percentage of the incorrectly clustered instancesis 56.785%. This is due to the fact of our significant dense datastructure. Nevertheless, we can still make some assumptionsbased on the centroids of the four clusters and the medianvalues of our full data set. For each cluster we discuss onlythe most important features that affect the build status. Thus,we define the symbols (+)/(+ +), (-)/(- -) that are placed insideour table as a positive/very positive or a negative/very negativecontribution accordingly. We skip to make conclusions fromboth gh lang and git num committers features because theirdata structure are far away from a normal distribution and theywould not provide any compelling insights.

1) PassedFor the passed cluster, our findings suggest thatbuilds that were committed on the master branch had themost positive contribution on the builds that passed. This wasexpected since developers are likely to be more aware andcareful when they commit on the master branch because ifthe build breaks, it would affect the whole project. Builds thatwere triggered as part of a pull request on GitHub, projects thathave a larger number of contributors are also a positive factorfor passed builds. That means that when a user make a requestof a new feature he is more cautious of errors that might leadto a build breakage. Also teams with more developers leadto more passed builds. High number of discussion commentsindicate a greater chance for a build to succeed. That mightbe because the developers discuss the issues that need to besolved during the development of a new project feature. Asexpected, the number of line changes in the production andthe testing code, and the number number of files modified,seems to be negative factor for passed builds. Also, the linesof executable production source code in the entire repositoryseems to have a positive correlation with the passed builds,which we can infer from that highly developed projects aremore likely to be successful. Test and assert density is also animportant negative factor for passed builds. Builds that includesmaller test and assert densities are expected to pass moreoften.

TABLE I: K-MEANS++ CLUSTERING MODEL

Attribute	Full Data	Errored	Canceled	Passed	Failed
-	(698998.0)	(100942.0)	(54079.0)	(346058.0)	(197919.0)
gh_is_pr	0.2327	0.2218	0.143 (-)	0.349 (+)	0.0594 (- -)
gh_lang	0.0774	0	1	0	0
git_branch	0.6384	0.6592	0.6217	1 (+ +)	0 (- -)
gh_team_size	32.7254	21.9662 (-)	7.8615 (- -)	37.2095 (+)	37.1661 (+)
gh_num_issue_comments	0.1731	0.2322 (+)	0.0693 (- -)	0.2481 (+)	0.0404 (- -)
gh_src_churn	70.5046	51.574 (-)	172.0885 (+ +)	64.0734 (-)	63.6479 (-)
gh_test_churn	40.1475	45.6711 (+)	60.1033 (+ +)	35.1098 (-)	40.6859
gh_files_added	0.6076	0.652	1.0223 (+)	0.5436	0.5833
gh_files_deleted	0.2668	0.1711 (-)	0.4433 (+)	0.2648	0.2709
gh_files_modified	4.4864	4.5409	6.6091 (+)	4.0546 (-)	4.6334
gh_tests_added	0.0097	0	0.1259 (+)	0	0
gh_tests_deleted	0.081	0	1.0466 (+)	0	0
gh_src_files	4.1202	3.9521	6.2653 (+)	3.8385	4.1124
gh_doc_files	0.2843	0.3798 (+)	0.1978	0.1946	0.416 (+)
gh_other_files	0.6654	0.5521	1.6696 (+)	0.5135	0.7143
gh_sloc	41180.3588	32954.7395 (-)	55533.3529 (+)	49471.1363 (+)	26957.495 (-)
gh_test_lines_per_kloc	3239.3559	4721.1781 (+)	533.0977 (- -)	2644.052 (-)	4263.9333 (+)
gh_test_cases_per_kloc	248.2759	289.1468 (+)	22.4602 (- -)	214.1243 (-)	348.8461 (+)
gh_asserts_cases_per_kloc	569.2683	618.8643 (+)	59.9723 (- -)	506.3662 (-)	793.1156 (+)
gh_by_core_team_member	0.8353	0 (- -)	0.7381	1	1
tr_duration	6297.1481	5690.7647	1491.9524 (- -)	6098.3105	8267.0402 (+)
tr_setup_time	4.6866	4.7813	6.3235 (+)	4.4506	4.6037
tr_tests_ok	2894.1142	3157.5545 (+)	1075.5916 (- -)	2915.295	3219.6103 (+)
tr_tests_fail	6.8543	8.3372 (+)	0.4693 (- -)	7.1721	7.2869
tr_tests_run	2900.9685	3165.8918 (+)	1076.0609 (- -)	2922.4672	3226.8972 (+)
tr_tests_skipped	31.4781	61.7747 (+)	3.2327 (- -)	36.6306	14.7351 (-)
tr_testduration	18921.9231	62878.2751 (+ +)	208.7979 (- -)	19788.7912	100.882 (- -)
git_num_committers	1.118	1.3129	1.1287	1.0826	1.0775
Incorrectly clustered instances:	396926 (56.785%)				

2) Failed: For the failed cluster, our findings suggest thatbuilds that were not triggered as part of a pull request,builds that were committed outside the master branch, buildsthat include less discussion comments, and builds with lessduration on the tests are all a strong positive contributionfor the builds that failed. This propose that developers tendto submit more often broken builds when their commitsare not pull requests, they are not committed at the masterbranch and their test duration is low. The team size and thechanged lines of production code seem to follow the samepattern as the passed builds, meaning that more developersand less changes on production code could lead to morefailed builds. Unexpectedly, the number of documentationsfiles changed seem to have be a positive factor for the failedbuilds. The more changes in the documentation files, the morethe possibilities for a build to break. Following the oppositepattern of the passed builds, the failed builds seems to benegatively affected by a large number of executable productionsource lines of code and small test and assert densities. Thatmakes us conclude that smaller projects with higher test andassert densities are more prone to failure. As expected, buildswith higher overall duration and with more tests passed andrun, are a positive factor for failed builds. That means thatbuilds with higher number of tests have more possibilities tofail. However, unexpectedly our findings also point out thatless tests skipped strangely lead to more failed builds.

3) Errored: For the errored cluster, we can see that testduration is the most positive factor. Tests with higher durationtend to get errored more. Also, whether the commit wasauthored by a core team member is a strong negative factor.All the builds that got errored were actually contributed byoutsiders. The team size has a negative contribution meaningless team members leads to an errored build. Following up, thenumber of issue comments has a positive contribution leadingto more errored builds when there are a lot of discussioncomments. Projects will less lines of production code changed,more lines of test code changed, less files deleted, and moredocumentation files changed are all factors for an erroredbuild. Same pattern as the failed tests is seen at the numberof executable production source lines and the test and assertdensities, making builds with less source lines changed andlarger test and assert densities more prone to errors. Lastly,the number of tests that run, passed, failed, and skipped, allof them affect positively the errored builds. The more we have,the more chances for a build to have errors.

4) Canceled: For the canceled cluster, we can observe thatmost of the builds that become canceled, they have much morestrong correlations on features than the other of our clusters.The lines of production code changed together with the linesof test code changed have a strong positive contribution on thetest that become canceled. That may be because big changeson the code can result to more code bugs and the developersmay start noticing them during the building time, thus theycancel the build before the build execution completes. On theother hand, test and assert densities have a strong negativecontribution, meaning that builds with less number of testcases and assertions per 1000 executable production sourcelines, are more likely to be canceled. Less overall buildduration, less tests run, passed, failed, or skipped, togetherwith less test duration show a strong impact on the canceledbuilds. This is expected since when a developer manually stopsthe building phase, a lot of the tests might remain untested.Moreover, canceled builds seem to have larger durations ofbuild setup time, leading to our understanding that developerstend to cancel the builds if they receive external delays. Finally,the more changes occur on all of the 3 types of files (generalfiles, test files, source files), the more likely is for a build tobe canceled. The latter suggests that if developers alter a bigamount of different files all together, they tend to cancel thebuild, maybe in order to rerun the execution in parts to be ableto trace errors and bugs more easily.

B. In-Depth Analysis

Do considerable changes in a project's churn and project's files affect the build status? When a developermake changes inside a project, we would expect that, themore the amount of code or files that he change, the morethe possibilities that he would introduce some kind of erroror bug. Table II contains the logistic regression model thatwe conducted in order to find some correlation between thefeatures and the build status. We performed a 10-fold stratifiedcross-validation in order to secure that our accuracy remainedat the same levels. The model shown has 75.9% accuracy andthe overall cross validation accuracy was 74%, which meansthat the model has a satisfying performance. However, the nullerror rate is exactly at the same percentage. That means thatour model could get 76% accuracy by predicting always thebuilds as passed.

Observing the coefficient values of our model, we can tellthat the most prominent predictors are the test churn, thenumber of files deleted & modified, and number of changedsource files & files that are neither production code nordocumentation. Our findings suggest that, when developersmake big changes on the test churn then it is more likely forthe build to pass. Files deleted, files modified, and the numberof source files changed, all three have negative coefficientswhich means that the more they increase then the less likelyis for a build to pass. This could be because developersaccidentally remove or alter files that are dependant for theproject to be functional. Moreover, big chances to the sourcecode might lead to more bugs. Lastly, an increment to thenumber of the files changed that are neither production codenor documentation, unexpectedly increases the possibilities for a build to pass.

Is test time, build time and build setup time a considerable variable that can produce more broken builds? Figure3 displays three different plots between overall build duration,test duration, and build setup time. Unfortunately, all threeplots fail to provide us with valuable information about whenbuilds break more often. All of our data is dense and mixed,so no accurate conclusions can be made. However, inside theplot of overall duration with test duration we can observe thatthere is a correlation of passed builds. Builds with an averageoverall duration of 25000 to 45000 seconds and with a small toaverage test duration of 300 to 2500 seconds tend to be moresuccessful than the others. But again, this does not provide uswith any important information about the connection betweenthe time variables and the build breakage.

TABLE II: PROJECT CHURN AND PROJECT FILES LOGISTIC REGRESSION MODEL

Variable	Coefficient
(Intercept)	1.153
scale(log(gh_src_churn + 0.5))	0.022 (8)
scale(log(gh_test_churn + 0.5))	0.115 (1)
scale(log(gh_files_added + 0.5))	-0.025 (7)
scale(log(gh_files_deleted + 0.5))	-0.104 (2)
scale(log(gh_files_modified + 0.5))	-0.065 (3)
scale(log(gh_tests_added + 0.5))	-0.004 (10)
scale(log(gh_tests_deleted + 0.5))	0.021 (9)
scale(log(gh_src_files + 0.5))	-0.058 (4)
scale(log(gh_doc_files + 0.5))	0.003 (11)
scale(log(gh_other_files + 0.5))	0.054 (5)
scale(gh_sloc)	-0.028 (6)
Accuracy	75.9%

IV. CONCLUSIONS

CI has been rising as a big success story in automated software engineering and that is why we need to take advantage of its aspects and further understand the patterns and the behaviors of their users. In this preliminary study, we tried to understand and reveal the factors that mostly affect the build status. Although, we reduced the initial data set size at a high degree in order to efficiently address our research questions, we still managed to come up with some valuable conclusions and topics for further study.

Through our k-means++ clustering model, we found out that passed builds occur more often when they are committed on the master branch. Failed builds mostly occur when they are not pull requests, they are committed on non-master branches, they have a low amount of issue comments and their test duration is short. Further on, errored builds have a massive amount of duration on their tests, and canceled builds tend to have small team size, small number of issue comments, and large changes on the source and test churn. Also they likely have small amount of tests run and their durations, which is trivial, since their execution is stopped earlier than the predetermined.

Our logistic regression models showed that large changes on projects' churn and files affect the build status in an surprising way. For example, the bigger the changes on the test churn, the more likely is for a build to pass. Moreover, we discovered that the larger the amount of tests run, the more the possibilities for a a build to be successful. We identified that builds that contain a high number of assert cases and a low number of test cases tend to break more often. Furthermore, time duration factor seems to be irrelevant to the build status. Regarding, the branches correlation with build status, we could only confirm that builds tend to be canceled less on the master branch. Finally, whether the commits were made by a core or a non-core developer affect the build breakage, could not be answered accurately through our data.

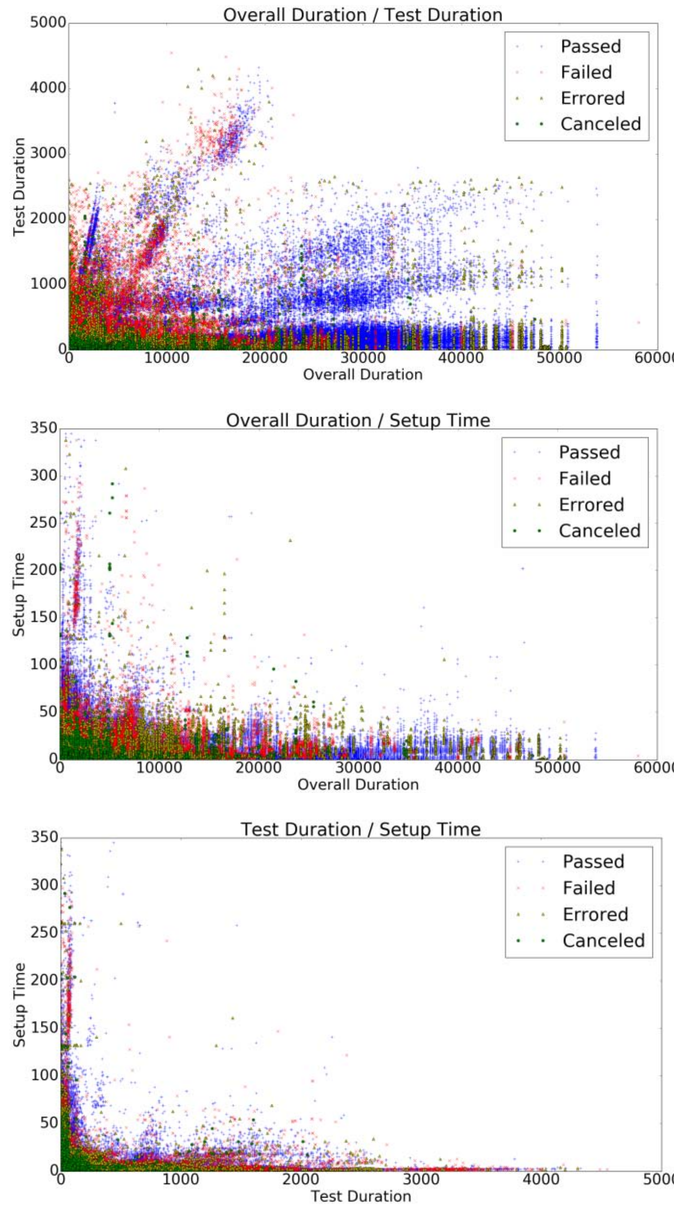

Fig. 3. Three 2D plots between the time-related features.

REFERENCES

[1] E. T. Barr, C. Bird, P. C. Rigby, A. Hindle, D. M. German, and*P. Devanbu, "Cohesive and isolated development with branches," in Proceedings Proceedings of the 15th International Conference Fundamental Approaches to Software Engineering (FASE),* 2012, pp. 316–331.

[2] M. Beller, G. Gousios, A. Panichella, and A. Zaidman, "When, how, andwhy developers (do not) test in their IDEs," in

Proceedings of the 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), 2015, pp. 179–190.

[3] M. Beller, G. Gousios, and A. Zaidman, "Oops, my tests broke the build:An analysis of Travis CI builds with GitHub," PeerJ Preprints, Tech.Rep., 2016. [Online]. Available: https://peerj.com/preprints/1984.pdf

[4] ——, "TravisTorrent: Synthesizing Travis CI and GitHub for full-stackresearch on continuous integration," in *Proceedings of the 14th International Conference on Mining Software Repositories (MSR)*, 2017, pp.1–4.

[5] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. German, andP. Devanbu, "The promises and perils of mining Git," in *Proceedings of the 2009 6th International Conference on Mining Software Repositories (MSR),* 2009, pp. 1–10.

[6] J. Cohen, P. Cohen, S. G. West, and L. S. Aiken, *Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences.* Routledge,2013.

[7] G. Gousios, "The GHTorent dataset and tool suite," in *Proceedings of the 10th International Conference on Mining Software Repositories (MSR),*2013, pp. 233–236.

[8] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H.Witten, "The WEKA data mining software: An update," *SIGKDD Exploration Newsletter,* vol. 11, pp. 10–18, 2009.

[9] L. Osterweil, "Software processes are software too," in *Proceedings of the 9th International Conference on Software Engineering (ICSE),* 1987,pp. 2–13.