On Association of Code Change Types and CI Build Failures in Software Repositories

***Abstract*—Building is an integral part of software development processes. Previous studies show that building systems break frequently during the software evolution, particularly in environments adopting continuous integration (CI) practices.**

To this end, we studied the impact of different change types, individually and together, on CI build failure rates. We compared the contribution of changes, which occurred due to various underlying reasons, including the addition of functional requirements, bug fixes, enhancement activities, and removing dependencies. The preliminary results showed adding new functionalities to software products contributed less to the CI failures than maintenance changes.

Moreover, we studied the characteristics of ultimate changes, with the purpose of identifying the common features among the change types, which contributed to failure. Finally, given the identified features, we trained a mathematical model to predict the failures according to the characteristics of the triggering change type. The model was able to accurately recognize the potential failure-inducing changes in the dataset with recall and precision of 78% and 53% relatively.

***Index Terms*—Mining software Repository, Continuous Integration, Build Failure**

I. Introduction

The four different types of software maintenance, including *corrective, preventive, perfective*, and *adaptive*, are each performed for different reasons and purposes throughout the software lifespan [1], [2]. Corrective maintenance is often due to bug reports, submitted by the users of the product, while preventative activities usually refer to the future so that your software can keep working as desired for as long as possible. Examples of preventive tasks include making necessary changes, upgrades, adaptations, and more. Preventative software maintenance address more negligible issues, which at a given time may lack significance, yet may turn into a large problem in near future. These are known as the software's latent faults which require to be detected and corrected to prevent their transformation into effective faults. Perfective changes often occur due to users need for new features or requirements, requested to be incorporated in the software to enhance the software into a potentially-best tool available for the users needs. Finally, Adaptive software maintenance addresses changing technologies as well as policies and rules regarding your software. For instance, changes in the operating system, cloud storage, or adjacent hardware components [2].

Software maintenance in an environment with continuous integration (CI) practices, is often performed by several developers, who frequently commit their code changes into a shared repository to be automatically integrated into a single project, tested, and eventually built [3].

In a CI process, a significant number of heterogeneous and potentially inconsistent changes are continuously integrated and built, and therefore a *build break* is the most common cause of failures [4], [5]. Once a building procedure breaks, the software development process will come to a halt, especially in collaborative and agile environments, where fixing the build suddenly becomes a top priority [6]–[8]. Build failures slow down the product's release pipeline, decrease team productivity, and increase the software production costs [9], [10]. In one study the researchers observed that total build failures of a project aggregate to a cost of more than 2,000 man-hours [11].

A commit build may break for several reasons, such as compilation errors or test failures. For obvious reasons, developers could check-in their code changes to the repository less frequently so that the overhead in the development process, caused by the build failures, is reduced. However, in this case, once the build breaks, troubleshooting and fixing the issues will be even more difficult as the code changes are larger and more complex, resulting in multiple conflicts in the system. Resolving these conflicts may eventually add a larger overhead to the process than repairing the build failures of smaller changes. Due to this reason, taking a pre-emptive action seems necessary to minimize the frequency of the build failures occurrence. The prediction of the issue types, which are more prone to generate build failures helps to take preventive actions in regard to the build failures.

To address the problem of build failure, several research groups aimed to identify the most influential underlying roots of build breaks [4], [6], [9], [10], [12], [13]. Some studies developed tools and plugins to support fixing build failures and make the recovery process faster and more efficient [14]–[16]. For instance, among them, Beller et al. [16] proposed TravisTorrent to process and analyze build reports specific to Travis CI, from GitHub commits and extract more information about the failures once occurred.

The ultimate goal is to minimize the CI failures through predicting the likelihood that a build may fail. This research intends to draw attention to the contribution of the underlying reasons for the change to the failures. This work, therefore, has two fold *SS: objectives?* . The first one aims to study the triggering issue types, requested by the end users, which primarily provoked the CI build failures. Moreover, since **addressing each issue type yields a different set of changes in the code**, the second fold of our research studies the characteristics of the ultimate code changes, independent of their underlying reasons. The second fold tends to identify the mutual features of the resultant failure-inducing code changes

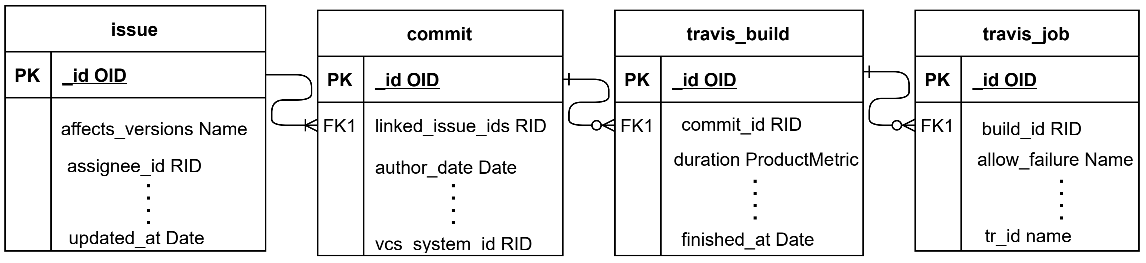

Fig. 1. Partial ER diagram of SmartSHARK database relevant to this work.

among all the user-requested issue types.

The reason is to further exploit the features of the user-reported issues, as well as their potential subsequent change impacts in the code to flag the potential failure-inducing user requests as they are posted and before their occurrence. For this, we trained multiple binary classifiers which assess the possibility of training a model for CI failure predictions according to the issues characteristics.

The prediction of CI build failure reveals, in advance, the critical issues which yield changes that are more prone to failure, once they are built in CI. Such warnings alert developers to prepare before the product is built and to take preventive measures. Therefore, the focus of this research will be on the issue types and issue-related metrics which contribute, and therefore allow to detect failure-inducing builds. For instance, are the perfective maintenance activities are more prone to CI build failures than the corrective changes? In particular, this work answers the research questions below:

- $\boldsymbol{RQ_1}$: Which change-requesting issues contributed more to CI build failures?
- $\boldsymbol{RQ_2}$: Are there common features among the changes which led to CI build failures?
- $\boldsymbol{RQ_3}$: Can we statistically predict the failure-inducing changes?

The first question, therefore, seeks for the change type which is more likely to result in CI build failure. For instance, does fixing bugs in the code or fixing code dependencies are more likely to trigger the CI build failures? How about code changes related to incorporating a new requirement or code improvement changes (e.g., refactoring)? Within each change type, once individually, and once again jointly, the second question studies the (common) features of those changes, which led to the failure. For instance, do changes, whose related issue was of *critical* priority more often led to the CI build failure, in comparison to the issues with *minor* priority? How about issues with the *re-opened* status versus those which have a *patch available*? The third question explores the possibility of building predictive models to predict the build failures according to the identified features in the previous research question.

II. Background

In this section, we provide the background knowledge required to better read the rest of the paper.

A. Travis CI

Continuous integration is a software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run. A build pipeline can consist of several tasks, but at a minimum three phases should be included:

1) Compiling code;
2) Executing tests, such as unit and integration tests; and
3) Deployment, which contains packaging the complied code into artifacts (*e.g.*, jar files) and deploying them.

Similarly, Travis CI provides a build environment for a repository, cloned into it, and then executes a set of build phases specified in a configuration file in .yml format. For instance, once linked to GitHub, Travis monitors the repository. Once a new pull request is opened, Travis receives a notification and executes the steps of the build pipeline as defined in its configuration file. If any of the build steps fails, the pipeline terminates and notifies users that the build is broken.

Travis includes multiple features which made the environment the developers' preferred option to start with build pipelines. For instance, Travis integrates with GitHub repositories, deploys to multiple cloud platforms, and supports different programming languages.

An overview of Travis build states is demonstrated in Figure 2. As shown, in Travis a build process may have one of the three final states:

- Build Errored: Once a build process breaks before the execution of build commands in *script* (e.g., due to dependency or dependency installation failures), the build is errored.
- Build Failed: Once build commands in Travis *script* fail or time out, the process still continues before the build is marked as failed.
- Build Passed: Once the execution of the entire build jobs is completed the status of the build is marked as successful.

In addition to the three status discussed above, canceled build status can occur in any phase and is triggered with an outside command.

B. SmartSHARK Dataset

For our analysis we used a publicly available dataset, namely SmartSHARK, release 2.2 [17], integrating a collection of various software projects and their evolving artifacts from publicly available sources like GitHub, Travis CI, JIRA

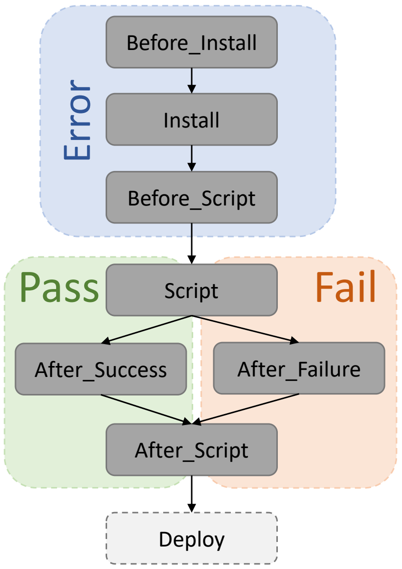

Fig. 2. Travis CI build process with final states of errored, passed, failed.

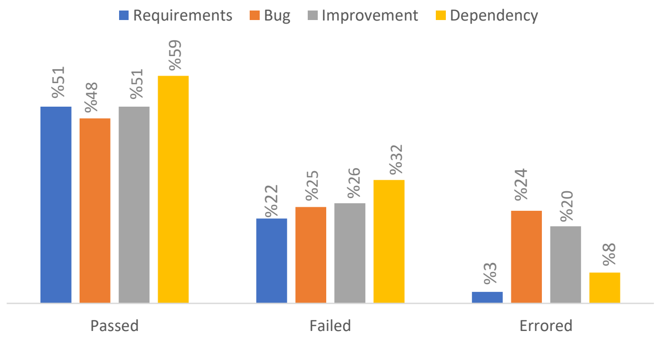

Fig. 3. Histogram showing the percentage of Failed, Passed, and errored verdicts for each issue type.

issue tracker [18] along with a number of analysis on the data. The dataset consists of 98 Apache project data, stored in MongoDb database, including 452,286 commits, 207,400 issues, 52,478 pull requests, and 89,160 Travis build records.

There are two versions of the dataset available, one being larger than the other version including code clone and multiple software metrics as well. We used the smaller dataset containing 36.3 GB of data, including the same information of the entire projects in the larger dataset except for the code and metrics, not being required in this work.

The portion of the database schema, which we used here, is shown in Figure 1 in the form of an E-R diagram. Each collection in the Figure contains several additional fields in the database, but due to space limitations, only a few of the fields and the relationship between the collections are demonstrated. The complete SmartSHARK schema definition along with their documentation is available in [19]. The artifacts of this research are publicly accessible on GitHub [1].

III. Whom to Blame in CI Build Failures?

This section provides a partial answer to RQ_1, while the following section provides statistical proofs for this research question.

We initially analyzed the available issue types in the collection, identifying 31 unique types of issues among the 98 projects. Among the issue types, *new functionality-*, *bugs-*, and *improvement-related* issues contained the largest number of records, including 13,216, 108,137, and 49,512 issues respectively. We additionally looked into three additional issue types to be included in the study, including *task*, *sub-task*, and issues related to *dependencies* consist of 11,367, 11,755, and 2,178 records in the database respectively. However, in our further manual analysis of several records for each issue type, we found significant inconsistencies among the types of issues labeled as "task" and "sub-task", leading us to exclude both these types from the study, while including the dependency-related issues.

As such the final types were selected from four major issue types, whose triggering reason rooted in explicitly-different classes of change. The first type includes the addition of new features to the software product (perfective maintenance), integrating new source code into the framework ("requirements"). The second is related to fixing product bugs (corrective maintenance), integrating corrected source code ("bug"). The purpose of the third type is to improve the product (preventive maintenance), integrating both new and corrected source code ("improvement"). Finally, the last group is related to maintaining the present dependencies ("dependency"), such as upgrading the libraries, which are in use by the software, and addressing the issues happened due to a change in frameworks (adaptive maintenance).

For the selected issues, the 'commit' collection, where all commit-related information is stored, was searched to identify the commits traced to each issue. As such, only a total of 5,286 issues were found to be traced to the commits, while the rest of the issue ids were not present in the collection. This could be due to the missing issue-commit trace links or due to not-yet-implemented new features. In either case, this study did not include the issues which were not linked to any commit in the database.

A total of 14,466 commits were retrieved through developing a query, searching the 'linked_issue_ids' field of the commit collection for the 'id' value of the 'issue' collection. Later for the selected commits, the 'Travis_build' collection of the schema was searched, resulting in the return of 901 builds linked to the commits, shown in Figure 1. The process of extracting the three artifact types, issues, commits, and Travis_build, took about 3 hours on a machine with Intel core i7 1.80GHz CPU with 16GB RAM.

The entire process was then automatically repeated, taking 24 hours for issues with type *'bug fix'*, 12 hours for type *'improvement'*, and 1 hour for type *'dependency'* tags. Table I shows the counts of the artifacts issues, traced issues, commits, and builds for each issue class. In case of dependency-related issues, we merged the two issue types of 'Dependency' and

[1]https://github.com/anonSubmissionGithub/MSR2023

'Dependency upgrade' as both refer to the same type of issue.

As shown in Table I, the number of ed issues related to bug fixes were significantly higher than those relevant to the new features, with a total number of 108,137, among which 49,583 were traceable in the commit collection. Since several issues were linked to more than one commit, there were 94,324 commits in total related to the bug fix issues. Among them, only 7,553 commits, relating to 4,693 issues, were traced in the Travis build records. This happens since the relevant commits, addressing the same concern, are often merged into a larger group of changes to be built at once, instead of re-building the entire system for each individual commit. The last three rows of the table show the number of Failed, Passed, and errored builds for each issue type. The total build number is larger than the value in the *build counts related to issues* in the sixth row since sometimes the unsuccessful builds were repeated multiple times before they succeed.

Figure 3 represents a histogram of the Failed, Passed, and errored build rates per issue for each change type. Builds with other verdicts, such as canceled, are not shown here. As shown, a trivial observation represents that the bug-fixing changes lead to the majority of the CI Failures, yet further analysis is required to draw any conclusion.

IV. RQ_1 Statistical Analysis Primary Change Type in CI Build Failures

To study the significance of our observations in the previous section, and to answer our first research question, we conducted a statistical analysis. As discussed earlier, RQ_1 attempts to study the significance and strength of the relation between the issue types and CI build final verdict.

A. Chi-square Test

Since the dataset majorly contains nominal data, in order to use the same statistical equation for all the variables, we converted the commits and builds counts from the count scale to the four nominal scales. This type of conversion results in type consistency within the analysis and therefore allows to measure and interpret the same metric for the entire variables in order to conduct a more fair comparison.

To assess the contingency between the nominal values, one commonly-used test is *Pearson Chi-square statistical test*, intended to test how likely it is that an observed distribution is due to chance [20]. In this study, we adopted Chi-square test to assess the significance of relation, as well as *Cramer's V metric* to measure the strength of the significant relations [21].

TABLE I
THE COUNTS OF *Commit and build artifacts* FOR EACH ISSUE TYPE.

	Reqrm.	**Bug**	**Imp.**	**Dpn.**
Total issues	13,216	108,137	49,512	1,798
Distinct issues in commits	5,286	49,583	23,758	318
Commits related to issues	14,466	94,324	52,646	584
Build count related to commits	901	7,553	4,731	37
Build count related to issues	471	4,693	2,793	25
Failed issue-related builds	**459**	**3,661**	**2,437**	**16**
Passed issue-related builds	**196**	**1,895**	**1,251**	**6**
errored issue-related builds	**223**	**1,785**	**948**	**0**

We applied *Pearson Chi-square test* to assess the independence of each variable pair with each other. The *V value* is one measure to interpret the relation between nominal values for the Chi-squared test. To measure the correlation, the V coefficient divides the square root of the Chi-squared statistic by the sample and the feature size.

In addition, Chi-square distribution is accompanied by a parameter called *degree of freedom (df)*, representing the number of independent variables within which the Chi distribution is calculated. *df* estimates the number of the variables in the calculation that are free to vary, while the rest of the vectors are constrained to lie in the samples Chi subspace [22]. The *df* can be considered as the minimum number of the vectors which is required to define the sample data in Chi subspace and can be calculated by subtracting the number of Chi-estimated parameters from the total number of values in the sample. A small *df* between an issue feature and the dependent variable (build verdict) determines that a smaller dimension is required to define the samples with respect to this feature. Furthermore, Pearson Chi-squared test requires a minimum expected count of observations to be satisfied in all cross-values of the two variables in the test. This means once a certain percentage of cross-value counts does not contain the Chi-expected number, then the test result is not valid for the pair. This does not necessarily rejects the correlation between the two variables, but rather states that more samples of a specific cross-value are required for the test to draw a conclusion. In the literature, once 50% or more of the cross-values counts are less than the Chi expected value, then the test is considered invalid [23], [24].

B. Discussion

To assess the significance of the observations in Figure 3, we need to show that the verdict differences we observed between the issue types and their build final states (*i.e.,* verdict) are statistically sufficient.

To represent the findings in Figure 3 are not due to chance and are, therefore, statistically significant, we conducted a Chi-square test. The final verdict of issues is considered as the dependent variable with values of *Failed* and *Passed* and the nominal type categories as the independent variable. The results of the test will determine whether or not there is a statistically significant relationship between Failures and change types. As shown in Figure **??** *SS: I am not sure which figure we are referring to* ,*SS: Is it table 2? we did not mention table 2* the significance value of 0.01, which is smaller than our selected value (0.05) verifies the existence of an association between the issue types and CI build verdicts.

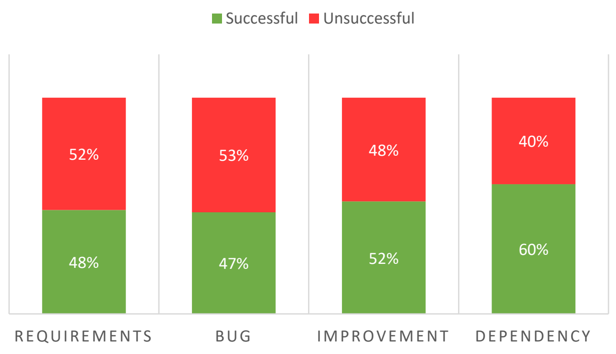

Fig. 4. Histogram showing the percentage of Failed and Passed verdicts for each issue type **after** merging the errored and canceled into the Passed class

> $\boldsymbol{RQ_1}$: The study showed that changing the existing source code for the purpose of resolving issues, **relevant to bugs**, lead to the largest percentage of the CI builds to be unsuccessful. The addition of the new functionalities, leading to the addition of the new source code, comes right after the bug-fixing changes. The Chi-square statistically verified the significance of the association between the issue types and CI build verdicts with $p - value < 0.05$. Please see Figures **??**, 3 and 5 for more details.

TABLE II
$\boldsymbol{RQ_1}$: COUNT AND CHI-SQUARE TEST RESULTS FOR CHANGE TYPES.

	Reqrm.	**Bug**	**Imp.**	**Dpn.**	**Total**
Verdict Failed	228	2,216	1,453	15	3,912
Verdict Pass	243	2,477	1,340	10	4,070
Total	471	4,693	2,793	25	7,982

	Value	**df**	**Asymptotic Significance**
Pearson Chi-Square	17.444	3	< .001
N of Valid Cases	7,982	-	-

V. RQ_2 STATISTICAL ANALYSIS PRIMARY FEATURES OF CHANGE IN CI BUILD FAILURES

This research question attempts to study whether a common feature potentially exists between the features of changes, which lead to the CI Failures. is common among all the issue types, and the final verdict.

A. Issues' Profiles

For conducting any analysis on issues, we initially need to build a dataset by creating a profile for each issue. We investigated the relations among a larger number of issues' characteristics than only the issue types, such as their priority and final status.

Thus, we ed a set of issue-related features which we hypothesized primarily contribute to the final state of the build process. For this, we first searched through issue collection in the database to scan and select the potentially correlated variables among the recorded features for issues present in the database. Among the 25 present features in the dataset, we selected *issue type, priority, status, resolution*, and counts of *commits* and *builds*, associated with each feature. These features and their existing values in the dataset are demonstrated in Table III, Accordingly, we then built a profile for each 7,982 issues in the four selected types.

Table III lists the selected independent variables and the dependent variable (i.e., CI final state) in the study. The possible values of each feature are presented in the row below. The first column contains a unique id of the issue. The second column shows the initially selected feature, issue types, with the values of requirements, bugs, improvements, and dependency issues. The next three columns contain the possible values for the issues' features of priority, status, and resolution. The status of an issue is limited to the five categories of blocker, critical, major, minor, and trivial, while the resolution feature provides a more detailed description of the problem's nature and takes 13 values, including *cannot reproduce, done, duplicate, fixed, implemented, information provided, invalid, none, not a bug, not a problem, resolved, won't fix, and works for me*. Column six and seven demonstrates the number of commits and builds associated with each issue. Finally, for the dependent variable, the build verdict, we initially searched the database for the values of *Failed, Passed, errored, canceled, created, and started*. However, the builds associated with the selected issue types did not have a final states of created or started.

As discussed earlier, several issues in the database were associated with more than one build entity, and therefore, contained more than one build verdict. For obvious reasons, in multiple cases, the verdicts were not necessarily consistent. For instance, the issues with Passed and errored builds, in the majority of the cases, were eventually followed by a successful build after the problem was resolved. Since the objective of the study is to study difficulties that occur during the software building, we only considered the two verdict values of *non-problematic* or *problematic* issues as Failed and Passed values respectively. As such, the issues with only Failed builds were considered non-problematic (Failed), while all other issues associated with any Passed, errored, or canceled builds were considered as problematic (Passed). As such, each category of change consisted of a relatively balanced data to train a binary classifier accordingly. ***SS:** As we can see from Figure 5* , the requirements contained 48% of data from class Failed and 52% from class Passed. Similarly, bug, improvement, and dependency types, respectively contained 47% Failed and 53% Passed records, 52% and 48%, and finally, 60% and 40%.

B. Cramer's Value

To identify the significant correlations between the issues and their build verdict, we measured a correlation metric for each feature showing the amount of their interference in the build's success or Failure. For this, we conducted a Chi-square test of the most contributing features of the change in the Failure of CI builds. In addition to measure the extent to which, the features contributed, we conducted a Chi test with Cramers V.

TABLE III
THE PROFILE (ID, INDEPENDENT, AND DEPENDENT VARIABLES) CONSTRUCTED FOR EACH ISSUE. THE ASSIGNED VALUES ARE LISTED UNDERNEATH EACH VARIABLE.

	Independent Variables						Depend.
Id	**Type**	**Priority**	**Status**	**Resolution**	**Commits**	**Builds**	**Verdict**
	Requirement, Bug, Improvement, Dependency	blocker, critical, major, minor, trivial	closed, in-progress, open, patch available, reopened, resolved	cannot-reproduce, done, duplicate, fixed, implemented, information-provided, invalid, none, not-a-bug, not-a-problem, won't-fix, workaround, works-for-me	$n = 1$, $2 \leq n \leq 3$, $3 < n \leq 10$, $n > 10$	$n = 1$, $2 \leq n \leq 3$, $3 < n \leq 10$, $n > 10$	Passed, Failed

The Cramer's V value measures the strength of a significant association between the two categories. The higher the value the stronger the correlation between the pair is. For instance, within the new requirements, commits counts has the highest contingency with the build verdict.

Moreover, Cramer's V value measures the strength of a significant association between the two categorical variables. It ranges between 0 and 1 Whenever the value is closer to 0, no association exists and whenever it is larger than .25, a strong relationship exists [25]. For instance, within the new requirements, commits counts have the highest contingency with the build verdict. Cramer's V can be defined using the following formula [26], where ϕ_c denotes Cramer's V, χ^2 denotes the Pearson chi-square statistic, N denotes the sample size, κ is the lesser number of categories of either variable.

$$\phi_c = \sqrt{\frac{\chi^2}{N(\kappa - 1)}} \tag{1}$$

C. Discussion

The study results identify which feature of the posted issues, regardless of the following change type in the source code, commonly contributed to the final verdict of the CI build. As shown, in Table IV the significance value of 0.001, which is a $p - value$ smaller than our chosen significance level ($\alpha = 0.05$) represents a significant correlation among the number of commits, number of builds, priority, and status type of the posted issues, with the CI build failure. However, the Cramer's value of 0.06 prevents us to scientifically draw a conclusion for the extent to which the issues' priority type contributed to the CI failures, since this value demonstrates a specially small and negligible association regardless of a significantly high confidence (significance value $<$ 0.001). This association is stronger between build counts and the CI failure rate with the Cramer's V values of 0.29 and 0.27 respectively, showing a large contribution, and the value of 0.12 is considered to be representative of a medium contribution between status type and failure.

The first column of Table IV represents the minimum count percentage of each test. The statistics which Passed to pertain to this assumption, and therefore were ignored in the study, are strikeout in the Table. This conveys that there is not enough samples of relation to draw statistically significant relation among the variables. However, the rest of the cells, with minimum count values of below and equal to 50%, are the features for which the Chi-value is valid because the minimum count assumption was met.

For the valid statistics then, the Chi value, (in column three table V) is compared to a calculated value from the Chi distribution table with the corresponding df and $p - value$. This value is called the *critical value* and is recorded in the sixth column. Once the Chi value is greater than the critical value, then the null hypothesis is rejected and an association can be inferred.

The test significance value finally determines whether the test was able to determine a statistically significant correlation between the two features or not.

To further investigate the occurrence of the same pattern individually in each category of change, we extended the study to each class of issue-related changes. The results are shown in Table V, representing the same pattern in changes relevant to the addition of new functionalities, as well as changes due to the improvement of the product's quality. In the change class relevant to removing dependencies, such pattern is not statistically shown, due to the large p-values. Yet, the alternative hypothesis is not rejected, and instead, it is solely concluded that there is not sufficient evidence to suggest an association between the variable and verdict in this category. The addition of samples in this category of change may improve the p-values so that a statistical conclusion can be drawn. For obvious reasons, the study shows that number of commits and builds of an issue has a *large* and positive association with the final build status.

$\boldsymbol{RQ_2}$: The study also shows that the **status** of the issue also has a *medium* positive association with the final build verdicts. This means that not only a large number of commits and builds often led to CI build failure (obviously), but the final status of the issue, as well, moderately contributed to identifying the verdict of the CI builds. Additionally, a *small* positive association is observed among the **priority** flags and the success of the build.

VI. RQ_3 Statistical Analysis Predicting CI Failure-Inducing Changes

The answer to RQ_2 statistically demonstrated that the current status of user-requested issues has an acceptable association with the final verdicts of CI builds. Issues with

TABLE IV

$\boldsymbol{RQ_2}$: The profile (Id, independent, and dependent variables) constructed for each issue. The assigned values are listed underneath each independent variable.

		Assumption	Pearson Chi-Square Tests				Cramer's V	
		minCount	Value	df	Significance	Critical	Value	Significance
All	commits-verdict	0.0%	597.46	3	<0.001	12.838	0.27	<0.001
	builds-verdict	0.0%	688.81	3	<0.001	12.838	0.29	<0.001
	priority-verdict	0.0%	28.86	4	<0.001	14.860	0.06	<0.001
	status-verdict	16.7%	125.51	5	<0.001	16.750	0.12	<0.001
	~~resolution-verdict~~	41.7%	16.05	11	0.139	5.578	0.04	**0.139**

TABLE V

$\boldsymbol{RQ_2}$: Chi-square Test Results for contribution of change attributes to CI build failure . For strikethrough results the number of visited samples were not enough to draw conclusion.

		Assumption	Pearson Chi-Square Tests				Cramer's V	
		minCount	Value	df	Significance	Critical	Value	Significance
Requirements	commits-verdict	0.0%	25.27	3	<0.001	12.838	0.23	<0.001
	builds-verdict	12.5%	20.40	3	<0.001	12.838	0.20	<0.001
	priority-verdict	40.0%	2.02	4	<0.001	14.860	0.06	<0.001
	status-verdict	40.0%	17.39	4	<0.002	14.860	0.19	0.002
	~~resolution-verdict~~	**71.4%**	10.59	6	0.102	18.548	0.15	0.10
Bugs	commits-verdict	0.0%	387.28	3	<0.001	12.838	0.28	<0.001
	builds-verdict	0.0%	464.91	3	<0.001	12.838	0.31	<0.001
	priority-verdict	0.0%	22.60	4	<0.001	14.860	0.06	<0.001
	status-verdict	50.0%	56.06	5	<0.001	16.750	0.10	<0.001
	~~resolution-verdict~~	**65%**	8.10	9	0.523	14.684	.042	0.523
Improvement	commits-verdict	0.0%	204.74	3	<0.001	12.838	0.27	<0.001
	builds-verdict	0.0%	201.109	3	<0.001	12.838	0.26	<0.001
	priority-verdict	0.0%	7.153	4	0.128	1.064	.051	0.12
	status-verdict	33.3%	66.071	4	<0.001	14.860	0.15	<.001
	~~resolution-verdict~~	**60%**	5.614	9	0.778	4.168	0.04	0.778
Dependency	~~commits-verdict~~	**66.7%**	1.681	2	0.432	0.211	0.25	0.432
	~~builds-verdict~~	**66.7%**	2.680	2	0.262	0.211	0.32	0.261
	~~priority-verdict~~	**66.7%**	0.104	2	0.949	0.211	0.06	0.949
	status-verdict	50%	3.175	1	0.075	2.706	0.35	0.075
	~~resolution-verdict~~	**66.7%**	.446	1	0.504	0.016	0.13	0.504

the current status of *in-progress*, *open*, *patch available*, and *reopened* are likely to be ed, while the code changes are happening and before the time that a build potentially Failed. Yet, the values of *closed* and *resolved* status are most likely not able until the build is complete. As such, this feature (issue's status) is shown useful for the prediction analysis of the CI build Failures, only when the implementation of the issue is not yet fully resolved.

For this reason, we further extended the study to measuring the extent to which the selected features of the Pass builds were associated with each other (rather than with the build Fails). This allows to identify the dependent variables whose values increase and decrease relative to each other.

A. Feature Interrelations of Failure-inducing Changes

Contingency coefficient provides a basic picture of the interrelation between two variables, statistically measuring the strength of the relationship between the relative changes of two nominal (categorical) variables.

To identify the presence of a latent relation among pairs of change features, we measured the dependency of each independent variable on the other variables to possibly replace non-predictable features with more predictable ones. The removal of such relations, additionally, removes the conditions' effects from our further analysis. Table VI displays the contingency coefficient matrix, where each cell demonstrates the value of contingency between each variable pair. Contingency matrix displays the (multivariate) frequency distribution of the variables, and the coefficient is calculated between 0 and 1, where a larger number is representative of a tighter association. As shown the *build-commit* and *resolution-status* pairs are highly correlated with a coefficient of 0.7. This size dependency shows that the value of the dependent variables are may affect the validity of the future model, accordingly trained.

As shown in Table VI, for obvious reasons a significant contingency is present between issue-related commits and builds count. In addition, the same significant contingency is demonstrated to be present between issues' status and their resolution.

The study shows a strong contingency among the **status and resolution** of the changes, as well as builds and commits counts. Such significant relationships between independent variables with respect to a certain dependent variable need to be removed before training a classifier on the data, to prevent biasing the predictive model.

TABLE VI
RQ_3: CONTINGENCY MATRIX

	#Commit	Priority	Status	Resolution	#Build
#Commit	1.00	0.11	0.119	0.08	0.70
Priority	0.01	1.00	0.122	0.08	0.08
Status	0.11	0.12	1.00	0.70	0.14
Resolution	0.08	0.08	**0.70**	1.00	0.11
#Build	**0.70**	0.08	0.14	0.12	1.00

B. Model Training

We investigated whether training a binary classifier allows to predict the failure-inducing issues. For a probabilistic classification, we applied discriminant function analysis (DFA). DFA is a statistical procedure that classifies unknown individuals and the probability of their classification into a certain group, assuming that the two different classes generate data based on different Gaussian distributions.

Since the objective is to predict the CI build failures before they occur, we selected the independent variables from the attributes of user-requested issues which are identifiable before the failure actually occurs. Yet, to provide a base line for comparison we assessed the trained model with the attributes, which were only predictable for a subset of values.

C. Discussion

The accuracy of this classification is reported in Table VII, using 10-fold cross-validation. For the comparison purposes, we report the results once using the identifiable attributes and once again with the attributes that are identifiable only for some values before the actual failure occurs but not the entire range of possible values. As shown, the top part of the table represents the selected dependent variable(s) and whether or not the variable(s) is(are) predictable ahead of time. The bottom part displays the count of the samples for each class of failed and passed. Considering the *Failed* samples as negative and *Passed* instances as positive instances, then the first green square from left represents the number of true positives as 3,166 (77.8%) records. These are the CI builds, which failed in reality and were correctly detected by the classifier as failure-inducing builds by only using the **priority** tag of the posted issues as the discriminating variable. The second green square, displays the number of true positives. As shown, 26.6% of the passed builds were correctly marked by the model. However, 2,870 successful builds were mistakenly marked by the classifier as potential to fail by the model, while they pass in real-life. While this represents 73.4% false positives, it lead to leave out marking only 22.2% of the actual failure-inducing builds.

Please note that in this problem, we are more interested in identifying the potentially failing builds, which will consequently delay the development process, rather than changes with a higher possibility to pass. This said, the model yields a higher chance of detecting CI build failures (recall of 77.78%) and a precision of about 52.45% .

Including resolution (predictable), the recall and precision are increased to 78.15% and 52.51% relatively.

RQ_3**:** With a reasonable amount of samples, we were able to statistically predict failure-inducing changes according to the user-posted issues' characteristics, with a recall of about 78% and precision of about 53%.

VII. Threats to validity

A threat to *construct validity* in our study is the potential bias caused by 98 software projects we used. We minimized this threat by selecting a dataset which contained projects relevant to a wide range of applications and functionalities. A threat to *internal validity* contains categorizing the issues based on the issue types. The issues could be assigned to a wrong category or mistakenly a wrong issue tag could be selected by the developers [27]. We only selected issue types with those tags which clearly mentioned the context of the change, to minimize this threat while this may lead to removing some issues that could fall under our selected categories. Another bias was raised because of the uneven sample size among different types of issues in the database. This could somehow influence the statistics we provided for the group containing all the issue types. We tried to apply a set of pre-processing techniques to minimize other statistical threats as much as possible as discussed in the paper. In addition, while some artifact counts were uneven among the four issue types, each type was individually balanced in terms of successful and failed build counts. The *external validity* includes the selection of open-source projects which might not be representative of or generalizable to closed-access commercial projects.

VIII. Related Works

A. Finding the Root Causes of Build Failures

Several prior studies have investigated the underlying causes of build breaks. A research on open source projects indicated projects build failures happen mainly due to unit testing failures [28]; a group of researchers found that builds generally fail because of failed test cases [12]; and similarly, a research study specified testing failures, compilation errors, and poor code quality as the most recurrent causes of build failures in Microsoft projects [4]. Another study identified poor code quality, identified with static analysis techniques, as the primary reason of failures [29]. While the primary focus of our research is on leveraging the issue types and other metrics from the issue tracker to predict failures, our final research question (RQ_5) analyses the root of build failures in CI Travis environments, identifying the unit test failures as the most

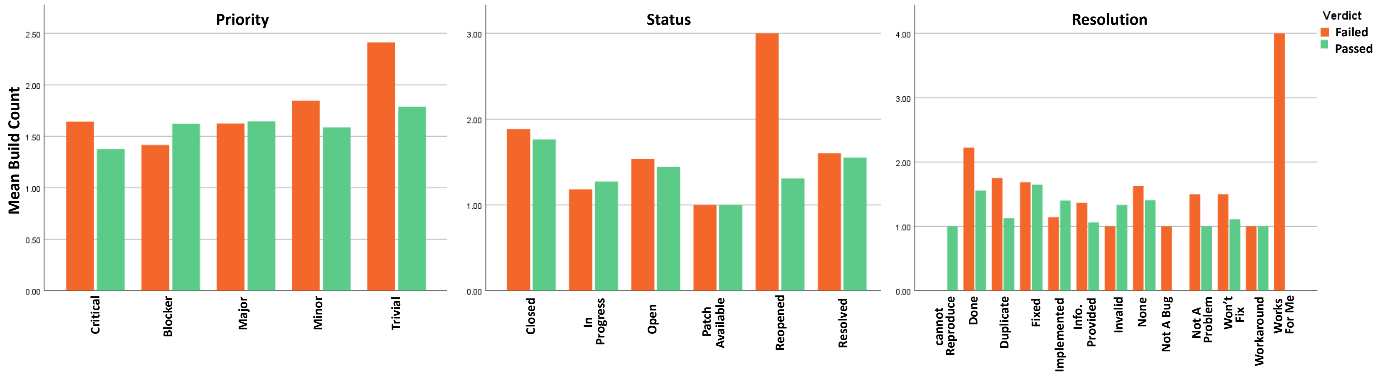

Fig. 5. Build Count for the independent variables, *Priority*, *Status*, and *Resolution*

TABLE VII
$\boldsymbol{RQ_3}$: DFA'S PREDICTION REULTS WITH POSSIBLE-TO-KNOW, NOT-POSSIBLE-TO-KNOW, BOTH, AND ALL DEPENDENT VARIABLES.

	Dependent Variable							
	Predictable				**Partially Predictable**			
	Priority		**Priority&Resolution**		**Priority&Status**		**All**	
Ground Truth	**Predicted Independent Value**							
	Failed	**Passed**	**Failed**	**Passed**	**Failed**	**Passed**	**Failed**	**Passed**
Failed	3,166 **(77.8%)**	904 (22.2%)	3,180 **(78.2%)**	889 (21.8%)	2,643 **(64.9%)**	1,427 (35.1%)	2,076 **(51.0%)**	1,993 (49.0%)
Passed	2,870 (73.4%)	1,042 **(26.6%)**	2,875 (73.5%)	1,037 **(26.5%)**	2,107 (53.9%)	1,805 **(46.1%)**	1,015 (25.9%)	2,897 **(74.1%)**

frequent initiation of failures in our dataset. *SS: We removed the why section. No RQ5 now*

B. Predicting/preventing Build Failures

Saidani et al. [10] proposed a search-based approach, a multi-objective genetic programming (MOGP) technique, to predict CI build failures. The model aims to identify the failures through finding the best combination of CI build features. Although the approach attempts to predict CI build failures, the primary focus is on generating a set of rules for the establishment of good practices by software developers.

Xia and Li [13] developed multiple classifiers to predict build failures for TravisTorrent projects, identifying the majority of the metrics to be specific to TravisTorrent database. The results indicated that the predictive models performed well for cross-validation scenarios but failed to perform well in on-line scenarios. Hassan and Wang [9] built a prediction model which performed with an average F-measure of 78% in cross-project prediction scenarios. Similarly, the majority of the metrics, which were selected to build the models were features of the TravisTorrent environment. The rest of the metrics were particularly defined at the method-level changes, such as method body change count and method signature change count. Rausch et al. [30] conducted an empirical study of CI builds only for Java-based applications. Their analysis had two folded directions. The first one intends to identify the types of errors in CI builds, while the second part aims to develop a general process and specific CI metrics for CI build failures

Compared to the above-mentioned works, this work searches for a set of generic features of the code changes and their initiating user requests, which more frequently lead to build failures. This research builds a predictive model, which features are not specific to a specific framework, and therefore, is generalizable to any environment.

C. Study/Analysis/Impact of CI Builds

Zhao et al. [31] performed a qualitative study to identify how the rise of CI practices changed the software development practices, in general. Several research studied and analyzed the impacts of CI practices on different software development approaches from multiple perspectives, such as code review [32], [33] and delivery time of pull requests [34]. The aforementioned research significantly contributed to better analyzing the software development environments which practice CI and in providing insight about the impacts of CI applications. Compared to these works, this research mainly focuses on predicting the final verdict of CI builds according to the characteristics of the precedent changes, resulting in taking preemptive actions to prevent the halt of the development process, if necessary. In addition to above-mentioned works, Zolfagharinia et al. [35] studied the CI build inflation by analyzing the relationship between Runtime Environments (RE) and Operating Systems(OS) and build failures on 30 million build records of CI environments. Their result indicates, the builds on Perl packages act differently on different Res and OSes. Paixao et al. [36] conducted a study to investigate the relationship between Non-functional requirements (NFR) and Travis-CI build statuses. However, they focused on NFR related build failures mainly where they categorized their types and duration to fix which is a different area of research.

D. Creating Developer Profile

Constructing and using profiles for software artifact is an active research area. One common application is to extract developer profiles from publicly available sources. For instance, multiple works created a profile for developers active on GitHub [37], [38]. Mining developer profile data is also used to match job advertisement [39], finding experts [40], [41], measuring developers contribution [42], for personalizing recommendations [43], identify the gender and nation wise diversity of the team [44], recommending relevant project [45], exploring the patterns of social behavior [46] and so on. Bao et al. [47] conducted a study to find long-time contributors for open-source applications based on their previous activities on Github. Tools like CVExplorer [48], CPDScorer [49] are also developed to mine technical skills by the developers. Xiong et al. [50] explored cross-site developer behavior on StackOverflow and GitHub T-graph analysis, LDA-based topics clustering, and cross-site tagging. Souza and Silva [51] analyzed Travis builds and their comments to see if negative sentiments have some impact on Travis builds. ***SS**: In contrast, based on the issue profile, our work focused on finding the potential relations between the issue types, consequent changes, and their features which are contributing more to the build failures to prevent build failures.*

E. Human Factors in CI Builds

Studies on impacts of human factors on successful build process are less common, but there has been recent progress. For instance, Wolf at al. studied the association of build fails with social factors of developers. Using social network analysis, they trained a model to predict whether an integration will fail based on its developers communications [52]. Other researchers studied features related to relationships between developers of build, such as communication, trust, and conflict. For instance, Phillips et al. found that social challenges of build team engineers impact their effectiveness and therefore, proposed to address social impediments in build teams in order to improve the build process [53].

Although mining data from software repositories, including Travis CI and Pull requests, is not a new research, creating issue profiles from the build reports has not yet been explored to the best of our knowledge. Since SmartSHARK [17] database integrates data from multiple sources, such as GitHub, Travis CI, Issue trackers, we used the data to study whether we can build issue profiles to later foresee the verdict of a CI build according to the nature of the change. In contrast, this work tends to emphasize the potential relations between the issue types, consequent changes, and their features which are contributing more to the build failures so that the potential failures are predicted and prevented in advance.

IX. Conclusion and Future Works

Continuous Integration (CI) is frequently practiced to provide the developers with the functionality to merge their code changes into a central repository to be automatically built and incorporated into the development process. Since building is an important part of the software development process, identifying the recurrent causes of build failure helps us to prevent build failure in the future. Having this in mind, we have analyzed the impact of several change types, including requirements-related changes, on CI build verdicts in 98 Apache projects. We are particularly interested in knowing whether the requirements changes have a more significant contribution to the build failures.

Comparing the requirements impacts with other major issue types, such as bug fixes and improvement activities, the results indicated a smaller percentage of build failures occurred after the requirements-relevant changes. However, our further statistical analysis revealed that the issue types do not have a statistically significant association with the CI failure rates. ***SS**: RQ1 result shows there is a relation* We additionally extracted and analyzed multiple other features of the issues, such as their priority and final status, to train a binary classifier to predict future CI build failures. While the results are encouraging but open to future improvements in the majority of issue types. Yet the model was able to detect the failure-inducing requirements with the highest accuracy of 75% among the rest.

In future, we plan to extract more software projects to include additional CI logs related to build job records. Extending our dataset, we intend to study dependencies among a larger set of attributes and change types.

References

[1] N. Chapin, J. E. Hale, K. M. Khan, J. F. Ramil, and W.-G. Tan, "Types of software evolution and software maintenance," *Journal of software maintenance and evolution: Research and Practice*, vol. 13, no. 1, pp. 3–30, 2001.

[2] "What is a software maintenance process? 4 types of software maintenance." [Online]. Available: https://cpl.thalesgroup.com/software-monetization/four-types-of-software-maintenance

[3] [Online]. Available: https://circleci.com/continuous-integration/

[4] A. Miller, "A hundred days of continuous integration," in *Agile 2008 conference*. IEEE, 2008, pp. 289–293.

[5] M. Hilton, N. Nelson, T. Tunnell, D. Marinov, and D. Dig, "Trade-offs in continuous integration: assurance, security, and flexibility," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 197–207.

[6] C. Vassallo, S. Proksch, T. Zemp, and H. C. Gall, "Every build you break: Developer-oriented assistance for build failure resolution," *Empirical Software Engineering*, vol. 25, no. 3, pp. 2218–2257, 2020.

[7] M. Sulír, M. Bačíková, M. Madeja, S. Chodarev, and J. Juhár, "Large-scale dataset of local java software build results," *Data*, vol. 5, no. 3, p. 86, 2020.

[8] Y. Lou, J. Chen, L. Zhang, D. Hao, and L. Zhang, "History-driven build failure fixing: how far are we?" in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019, pp. 43–54.

[9] F. Hassan and X. Wang, "Change-aware build prediction model for stall avoidance in continuous integration," in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2017, pp. 157–162.

[10] I. Saidani, A. Ouni, M. Chouchen, and M. W. Mkaouer, "Predicting continuous integration build failures using evolutionary search," *Information and Software Technology*, vol. 128, p. 106392, 2020.

[11] N. Kerzazi, F. Khomh, and B. Adams, "Why do automated builds break? an empirical study," in *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2014, pp. 41–50.

[12] 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)*. IEEE, 2017, pp. 356–367.

[13] J. Xia and Y. Li, "Could we predict the result of a continuous integration build? an empirical study," in *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 2017, pp. 311–315.

[14] "Log parser plugin," https://wiki.jenkins.io/display/JENKINS/Log+Parser+Plugin, accessed: 2022-01-28.

[15] C. Macho, S. McIntosh, and M. Pinzger, "Automatically repairing dependency-related build breakage," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2018, pp. 106–117.

[16] 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)*. IEEE, 2017, pp. 447–450.

[17] A. Trautsch, F. Trautsch, and S. Herbold, "Msr mining challenge: The smartshark repository data," in *Proceedings of the International Conference on Mining Software Repositories (MSR 2022)*, 2021.

[18] [Online]. Available: https://issues.apache.org/jira/secure/Dashboard.jspa

[19] [Online]. Available: https://smartshark2.informatik.uni-goettingen.de/documentation/

[20] K. P. F.R.S., "X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 50, no. 302, pp. 157–175, 1900. [Online]. Available: https://doi.org/10.1080/14786440009463897

[21] H. Cramer, "Mathematical methods of statistics (princeton: Princeton universitypress, 1946)," *CramérMathematical Methods of Statistics1946*, 1946.

[22] M. L. McHugh, "The chi-square test of independence," *Biochemia medica*, vol. 23, no. 2, pp. 143–149, 2013.

[23] S. Onchiri, "Conceptual model on application of chi-square test in education and social sciences," *Educational Research and Reviews*, vol. 8, no. 15, pp. 1231–1241, 2013.

[24] D. Sharpe, "Chi-square test is statistically significant: Now what?" *Practical Assessment, Research, and Evaluation*, vol. 20, no. 1, p. 8, 2015.

[25] [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2452247318302164

[26] [Online]. Available: https://www.spss-tutorials.com/cramers-v-what-and-why

[27] S. Herbold, A. Trautsch, and F. Trautsch, "Issues with SZZ: an empirical assessment of the state of practice of defect prediction data collection," *CoRR*, vol. abs/1911.08938, 2019. [Online]. Available: http://arxiv.org/abs/1911.08938

[28] C. Vassallo, G. Schermann, F. Zampetti, D. Romano, P. Leitner, A. Zaidman, M. Di Penta, and S. Panichella, "A tale of ci build failures: An open source and a financial organization perspective," in *2017 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 2017, pp. 183–193.

[29] F. Zampetti, S. Scalabrino, R. Oliveto, G. Canfora, and M. Di Penta, "How open source projects use static code analysis tools in continuous integration pipelines," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 334–344.

[30] T. Rausch, W. Hummer, P. Leitner, and S. Schulte, "An empirical analysis of build failures in the continuous integration workflows of java-based open-source software," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 345–355.

[31] Y. Zhao, A. Serebrenik, Y. Zhou, V. Filkov, and B. Vasilescu, "The impact of continuous integration on other software development practices: a large-scale empirical study," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 60–71.

[32] M. M. Rahman and C. K. Roy, "Impact of continuous integration on code reviews," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 499–502.

[33] N. Cassee, B. Vasilescu, and A. Serebrenik, "The silent helper: the impact of continuous integration on code reviews," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2020, pp. 423–434.

[34] J. H. Bernardo, D. A. da Costa, and U. Kulesza, "Studying the impact of adopting continuous integration on the delivery time of pull requests," in *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. IEEE, 2018, pp. 131–141.

[35] M. Zolfagharinia, B. Adams, and Y.-G. Guéhéneuc, "A study of build inflation in 30 million cpan builds on 13 perl versions and 10 operating systems," *Empirical Software Engineering*, vol. 24, no. 6, pp. 3933–3971, 2019.

[36] K. V. Paixão, C. Z. Felício, F. M. Delfim, and M. d. A. Maia, "On the interplay between non-functional requirements and builds on continuous integration," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 479–482.

[37] J. E. Montandon, L. L. Silva, and M. T. Valente, "Identifying experts in software libraries and frameworks among github users," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 276–287.

[38] A. Santos, M. Souza, J. Oliveira, and E. Figueiredo, "Mining software repositories to identify library experts," in *Proceedings of the VII Brazilian Symposium on Software Components, Architectures, and Reuse*, 2018, pp. 83–91.

[39] C. Hauff and G. Gousios, "Matching github developer profiles to job advertisements," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 362–366.

[40] A. Mockus and J. D. Herbsleb, "Expertise browser: a quantitative approach to identifying expertise," in *Proceedings of the 24th international conference on software engineering. icse 2002*. IEEE, 2002, pp. 503–512.

[41] D. Schuler and T. Zimmermann, "Mining usage expertise from version archives," in *Proceedings of the 2008 international working conference on Mining software repositories*, 2008, pp. 121–124.

[42] G. Gousios, E. Kalliamvakou, and D. Spinellis, "Measuring developer contribution from software repository data," in *Proceedings of the 2008 international working conference on Mining software repositories*, 2008, pp. 129–132.

[43] A. T. Ying and M. P. Robillard, "Developer profiles for recommendation systems," in *Recommendation Systems in Software Engineering*. Springer, 2014, pp. 199–222.

[44] M. Ortu, G. Destefanis, S. Counsell, S. Swift, R. Tonelli, and M. Marchesi, "How diverse is your team? investigating gender and nationality diversity in github teams," *Journal of Software Engineering Research and Development*, vol. 5, no. 1, pp. 1–18, 2017.

[45] M. Guendouz, A. Amine, and R. M. Hamou, "Recommending relevant open source projects on github using a collaborative-filtering technique," *International Journal of Open Source Software and Processes (IJOSSP)*, vol. 6, no. 1, pp. 1–16, 2015.

[46] Y. Yu, G. Yin, H. Wang, and T. Wang, "Exploring the patterns of social behavior in github," in *Proceedings of the 1st international workshop on crowd-based software development methods and technologies*, 2014, pp. 31–36.

[47] L. Bao, X. Xia, D. Lo, and G. C. Murphy, "A large scale study of long-time contributor prediction for github projects," *IEEE Transactions on Software Engineering*, 2019.

[48] G. J. Greene and B. Fischer, "Cvexplorer: Identifying candidate developers by mining and exploring their open source contributions," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, 2016, pp. 804–809.

[49] W. Huang, W. Mo, B. Shen, Y. Yang, and N. Li, "Cpdscorer: Modeling and evaluating developer programming ability across software communities." in *SEKE*, 2016, pp. 87–92.

[50] Y. Xiong, Z. Meng, B. Shen, and W. Yin, "Mining developer behavior across github and stackoverflow." in *SEKE*, 2017, pp. 578–583.

[51] R. Souza and B. Silva, "Sentiment analysis of travis ci builds," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 459–462.

[52] T. Wolf, A. Schroter, D. Damian, and T. Nguyen, "Predicting build failures using social network analysis on developer communication," in *2009 IEEE 31st International Conference on Software Engineering*. IEEE, 2009, pp. 1–11.

[53] S. Phillips, T. Zimmermann, and C. Bird, "Understanding and improving software build teams," in *Proceedings of the 36th international conference on software engineering*, 2014, pp. 735–744.