The Impact of Test Ownership and Team Structure on the Reliability and Effectiveness of Quality Test Runs

Kim Herzig
Microsoft Research
Cambridge, UK
kimh@microsoft.com

Nachiappan Nagappan
Microsoft Research
Redmond, USA
nachin@microsoft.com

ABSTRACT

Context: Software testing is a crucial step in most software development processes. Testing software is a key component to manage and assess the risk of shipping quality products to customers. But testing is also an expensive process and changes to the system need to be tested thoroughly which may take time. Thus, the quality of a software product depends on the quality of its underlying testing process and on the effectiveness and reliability of individual test cases.

Goal: In this paper, we investigate the impact of the organizational structure of test owners on the reliability and effectiveness of the corresponding test cases. Prior empirical research on organizational structure has focused only on developer activity. We expand the scope of empirical knowledge by assessing the impact of organizational structure on testing activities.

Method: We performed an empirical study on the Windows build verification test suites (BVT) and relate effectiveness and reliability measures of each test run to the complexity and size of the organizational sub-structure that enclose all owners of test cases executed.

Results: Our results show, that organizational structure impacts both test effectiveness and test execution reliability. We are also able to predict effectiveness and reliability with fairly high precision and recall values.

Conclusion: We suggest to review test suites with respect to their organizational composition. As indicated by the results of this study, this would increase the effectiveness and reliability, development speed and developer satisfaction.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Software Metrics—Process metrics

General Terms

Management, Measurement, Reliability, Human Factors.

Keywords

Empirical software engineering, organizational structure, software testing, reliability, effectiveness

ESEM'14, September 18–19, 2014, Torino, Italy.

1. INTRODUCTION

Testing is part of a software engineer's daily development process. Changes to a new or existing system should be tested to ensure that the changed code base matches specifications and works as desired. As such, testing is a crucial development step. However, daily tests may slow down product development: testing large and complex software systems can easily take hours and testing each and every code change can easily become a time and resource demanding operation. Therefore, frequently executed test cases should be of high quality. Ineffective or unreliable tests may use expensive resources ineffectively and slow down product development without adding essential benefit—or may even harm the product by letting code issues slip into the final product. As a consequence, test suites have a direct impact on development speed and product quality. From a test engineer's perspective, writing and maintaining high quality test cases and test suites can be time consuming and difficult, especially if the underlying product changes frequently or drastically. That may be particularly the case for test suites combining test cases of multiple authors. Such test suites might require test authors to sync their changes to ensure that changed test cases do not impact other test cases executed in the same test suite. With respect to shared code ownership, earlier studies have shown that the organizational structure of software developers can influence code quality [1,2,3]: e.g. code entities with distributed code ownership tend to be more defect-prone.

In this paper, we investigate the impact of organizational structure on the effectiveness and reliability of test executions. Do test suites owned by a smaller part of the development organization perform better when compared to test suites whose owners are distributed over a wider range of the organizational structure? To answer this question, we performed an empirical study on Microsoft Windows build verification (BVT) test suites. We related effectiveness and reliability measures of each test suite to the complexity and size of the organizational sub-structure that enclose all owners of test cases executed by the corresponding test suite. More specifically, using organizational metrics, we are able to build prediction models to

- Identify above-median effective test suites with precision and recall values around 0.7.
- Identify test suites with below-median reliability (number of false failures) with precision values around 0.8 and recall values around 0.9.
- Our results show, that organizational structure impacts both test effectiveness and test execution reliability.

The organization of this paper is as follows. Section 2 provides an overview of the Microsoft Windows build verification test system. Section 0 contains a description on how we measured organizational structure while Section 4 contains details on our approach to measure test effectiveness and reliability. Section 5 outlines our experimental setup. The results of this study are given in Section 6. Section 7 describes related work while Section 8

discusses threats to validity. We close with Section 9 giving a summary and discussing implications of our findings.

2. WINDOWS QUALITY TEST SUITES

The development process and the branching system of Windows are organized around a branching tree as shown in Figure 1. Each vertex of the tree represents a code branch, directed edges between these vertices correspond to integration paths. Developers submit their code changes to development branches (leaf nodes) and let these code changes merge and integrate against each other using so called integration branches (inner vertices) until the code changes are merged into the root node of the tree, which corresponds to the stable trunk branch holding the current stable version of Windows.

Each edge of the branching tree is guarded by a *quality test suite*—a set of system and integration tests (Figure 2). Quality test suites ensure that code changes being merged into a branch are of high quality. These quality test suites automatically execute whenever developers schedule an integration request between branches. Consequently, code changes "travelling" from development branches to the trunk branch have to pass multiple of these quality test suites—at least once on every branch level. Thus, quality test suites are executed multiple times in the daily development process and have a direct impact on product development. For a more detailed description of the Windows branching and quality test suite system, we refer to Bird and Zimmermann [4].

In this study, we concentrate on Windows build verification (BVT) quality test suites. BVT suites ensure the integrity of the current Windows code base and the basic its functionality. BVT test suites remain execute the same test content on every branch level.

2.1 Quality Test Suite Composition

A single quality test suite contains multiple test cases (so called *test runs)*. Each test case executes a series of test steps. Each test step is owned by an engineer that contributed the test suite to the set of quality tests. As a consequence, test suites might execute test steps owned by different engineers that might belong to different organizational subgroups and development teams.

The test team is responsible for the composition of the individual test suites, test framework, and the triage of test failures. The testers themselves contribute test cases but not all BVT test cases are contributed by testers.

Test cases are independent, except for general setup procedures that fetch and installs the current Windows binaries in a test bed. A test case has no effect on later executed tests and can be treated as self-contained and independent.

3. ORGANIZATIONAL STRUCTURE

Similar to the branching tree (Figure 1), the organizational structure of development teams can be represented as an organizational tree. Each vertex of the organizational tree corresponds to a person. The children of an organizational tree vertex correspond to those persons directly reporting into the person represented by the current tree vertex. The root node of this organizational tree is the CEO of the company. Similar to branches, we can also organize managers into management levels. The CEO of the company would be manager level zero, managers directly reporting into her would be level one managers, etc. A more detailed discussion is presented by Nagappan, Murphy and Basili [5].

As discussed in Section 2.1, test content can be contributed from engineers all over the organizational structure and we can assign organizational tree vertices to test content owned by the corresponding engineer or manager. Using the association between

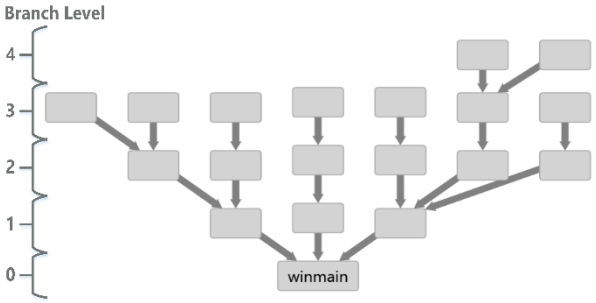

Figure 1: Windows branching tree hierarchy.

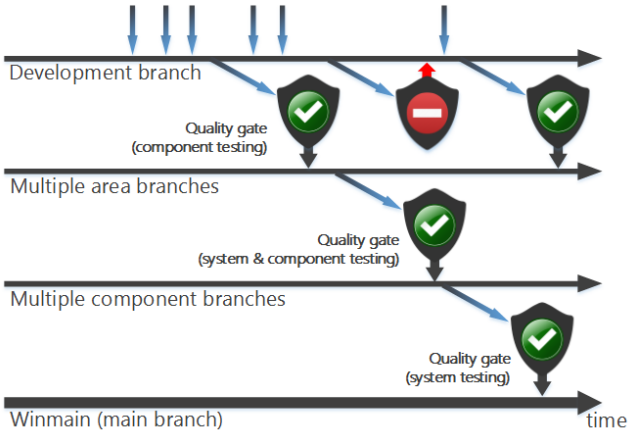

Figure 2: Code changes have to pass quality test suites to get integrated into lower level branches. This figure represents a vertical slice through branch tree shown in Figure 1.

organizational tree structure and test content, we can define tree based metrics of organizational structure for test cases and test suites based on test content ownership. The organizational tree can be interpreted as a description of official communication paths and responsibilities and thus reflects the distance between two organizational sub-trees. Two engineers reporting into the same manager are likely to have daily personal contact compared to engineers having no common direct manager. Overall, we suspect that the number of engineers contributing test content and their organizational dependencies impact the quality of test suites. A high number of contributing test owners might also increases the diversity of test cases. If contributors of a test suite span multiple organizational sub-trees, the higher the likelihood of long communication channels, which may impact test effectiveness and test reliability.

3.1 Organizational Metrics

A summary of the organizational metrics used in this study is given in Table 1. We can group these metrics into four main groups.

Number of contributors

This group of metrics simply counts the number of engineers that contributed to a test suite (*NumOwner*). The higher the number of distinct test owners of a test suite the higher might be the diversity and size of the test. However, having more people contributing to a test suite requires communication efforts and may cause the overall test suite to be less focused and more fragile.

Test owners that no longer belong to the organizational structure (e.g. left the company) are treated separately. The metrics *NumOwnerLeftOrg* and *PCOwnerLeftOrg* represent the absolute

Table 1: Organizational test suite metrics.

Metric name	Short description
Number of contributors	
NumOwner	The number of distinct engineers that own at least one test in the test suite.
NumOwnerLeftOrg	The number of distinct engineers that own at least one test in the test suite and that are no longer in the organizational structure of Microsoft (left the company).
PCOwnerLeftOg	The relative number of distinct developers that own at least one test case in the test suite but are no longer part of the organizational structure: $NumOnwerLeftCompony/NumOwner$.
Number of managers	
NumL2	The number of level two (L2) managers whose corresponding sub-trees contain all engineers that own at least one test case of the test suite.
NumL3	The number of level three (L3) managers whose corresponding sub-trees contain all engineers that own at least one test case of the test suite.
NumL4	The number of level four (L4) managers whose corresponding sub-trees contain all engineers that own at least one test case of the test suite.
NumL5	The number of level five (L5) managers whose corresponding sub-trees contain all engineers that own at least one test case of the test suite.
HighestCommonManager	The highest manager level (L3 > L2) whose sub-organization contains all engineers that own at least one test case of the test suite.
Organizational communication paths	
LongestDevDistance	The longest distance between two developers in the organizational sub-tree spanned by test contributors. The distance between two developers is measured by the length of the shortest path between the two corresponding organizational sub-tree vertices.
MedianDevDistance	The median of distances (length of shortest paths) between all pairs of developers contributing test content to a test suite.
MeanDevDistance	The mean of distances (length of shortest paths) between all pairs of developers contributing test content to a test suite.
MeanPathLength	The mean length of all paths between test owners.
SizeOfLargestClique	Number of vertices of organizational sub-graph such that any two test owners are connected via a single edge.
Number of aliases	
NumAliasGroups	The number of distinct mailing lists associated with test cases of the test suite. Does not include the number of system aliases (*NumSystemAlias*).
MedianAliasGroupSize	The median number of mailing list members referenced by mailing lists registered as test owners.
NumSystemAlias	The number of distinct aliases that point to a system alias—not to a mail distribution list. These aliases are real system accounts.

and relative number of test owners that contributed to a test suite but are no longer part of the organizational structure. Tests owned by engineers no longer associated with the organization are likely to be outdated and might cause test results to be less reliable. On the other hand, these test might also provide significant value by documenting knowledge on code behavior and code properties no longer present in the development team and which would be lost when removing the test.

Number of managers

Metrics simply counting the number of (distinct) test owners disregard the dependency between these owners—are the owners in the same organizational structure or do these owners belong to completely different organizational sub-trees? Engineers associated with two different teams will have different management chains (vertices on path to root node of tree). Engineers with the exact same manager chain belong to the same team. The distance between two development teams corresponds to the length of the common management chain. The shorter the common management chain, the longer the distance (analogue to distances between files in a directory tree).

The metric *NumL2* counts the number of distinct level two managers that appear in management chains of all test owners contributing to a test suite. A metric value of one means that all test owners of the corresponding test suite report into managers below the same level two manager. A metric value larger than one would indicate that test owners belong to very different organizational sub-groups. A metrics value of zero would correspond to a scenario in which one of the higher manager (L1 or L0) own the entire test content—a rather unlikely scenario. We compute the metrics *NumL3*, *NumL4*, and *NumL5* analogue.

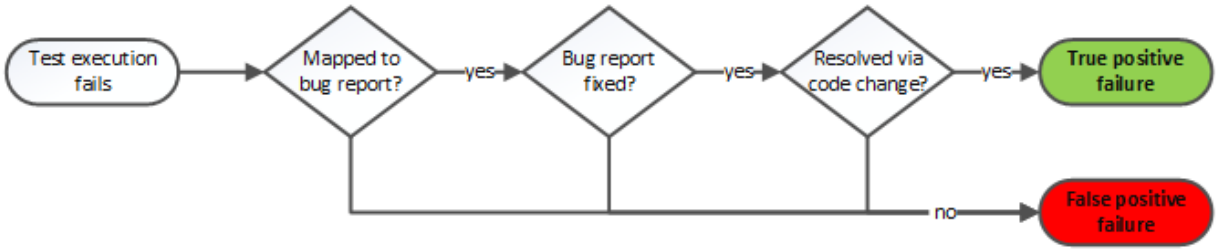

Figure 3: Flow chart describing the process to separate test failures reporting code issues from test executions failing due to other reasons than code issues (e.g. test and infrastructure issues).

Organizational Communication Paths

Engineers contributing to the same entity (in our case a test suite) are likely to impact each other and thus might have to communicate about issues, and strategies. The fact that engineers are distributed across different development teams might impact speed and ability to communication with each other or the cooperativeness in general. To capture such problems, we compute a set of metrics measuring the length of organizational communication paths between test owners. To measure the length of "official" communication channels between test owners, we compute the distance between two test owners using the length of the shortest path between the two corresponding organizational tree vertices. It is very likely that communication channels do not match these organizational communication paths, but longer management paths may very well indicate longer and less reliable communication channels between engineers. We use the distance (length of the shortest path) between two engineers in the organizational tree as an indication for communication quality. For each test suite, we measure all organizational distances between developer pairs and aggregate these distance measurements using three aggregations: max (*LongestDevDistance*), median (*MedianDevDistance*), and mean (*MeanDevDistance*). We also measure the average path length of all paths between all test owners (*MeanPathLength*).

To determine the size of compact and efficient communication subsets of test owners, we compute cliques of test owners—subset of test owners in which every two test owners are directly connected to each other—and use the maximal size of these cliques for each test suite (*SizeOfLargestClique*). A test suite for which this metric equals the number of test owners would mean that all owners directly report into each other—a rather unlikely scenario.

Number of aliases

Instead of single humans, tests can also be associated with aliases—user accounts that do not represent a human but rather point to a mailing list or a system account. In such cases, we resolved the corresponding email aliases to individual engineers and add these engineers as owners instead of the original alias.

However, the fact that there exist mailing lists for individual test cases means that the owners of such test cases are well organized and take shared responsibility for the test content and thus are also likely to communicate over fast channels not following official management structures. For each test suite, we count the number of distinct mailing lists marked as test owners for at least one test case of the corresponding test suite (*NumAliasGroups*). To reflect the size of these mailing lists, we report the median number of distinct human members of these mailing lists (*MedianAliasGroupSize)*.

For certain test cases, the test owner is pointing to a system alias. Different to mailing lists, these aliases cannot easily be resolved and we could not identify individual engineers that own the test content or sign responsible for it. The separate measurement *NumSystemAlias* reflects the number of distinct system aliases that own at least one test case of the corresponding test suite.

As mentioned in Section 2, this study is carried out on build verification (BVT) test suites only. The integration level and number of test steps of these tests remains the same for all branch levels. Thus, organizational test metrics do not depend on the branch level a test is executed on.

4. MEASURING TEST QUALITY

The quality of a test suite or test case can be defined in different ways, depending on the perspective and current problem domain. In this study, we are interested in two different but related test quality properties: test suite *effectiveness* and test suite *reliability*.

4.1 True and False Test Failures

Test cases can either fail or pass. In an ideal world, a failing test case would indicate a code issue that needs to be fixed. In reality, a test might fail due to many different reasons and not all test failures relate to code issues. At the level of system and integration tests, failing test cases can also be due to test and infrastructure issues; e.g. a test fetching a file from a remote server can fail if the remote server is currently not available. Although the test fails, there is no reason to believe that the tested code base contains a code issue that needs to be fixed. We call test failures due to other reasons than code issues (mostly test and infrastructure issues) *false test failures*. Analogue, we call test failures due to code issues *true test failures*.

To measure the quality of a test case, it is essential to differentiate between false and true failures. Test cases reporting many true failures are effective as they detect and prevent bugs from being shipped to the customer. Test cases frequently reporting false failures are considered not reliable. Furthermore, each test failure requires a manual failure triage. As a consequence, test cases reporting false test failures add high cost to the development process by triggering human effort to triage the false failure. In order to separate true from false test failures, we trace development activities that occurred after a test failure (see Figure 3):

1. First, we check whether the test failure is associated with a bug report. Test failures not associated with bug reports are likely to be false test failures. Test failures get triaged by a test failure triage team before being assigned to engineers. The fact that two teams look at the failure make bug reports the main communication channel. Thus, test failures reporting code issues but not being associated with bug reports are rather unlikely.
2. Next, we check whether the bug report got marked as *resolved* and *fixed*. Some false test failures are documented using bug reports. However, these bug reports get rarely marked as fixed, and if so, do not trigger a source code change (see next step).
3. Only if a test failure is associated with a bug report that was resolved by submitting a code change to the code base, we

mark a test failure a *true test failure*. To check for code changes associate with bug reports, we use the CODEMINE [6] infrastructure that checks commit messages for bug report references and bug reports for commit references.

4.2 Test Suite Effectiveness

There exist multiple ways to measure the effectiveness of tests but in the context of this paper, we are particularly interested in tests that find actual code issues. The main purpose of tests is to detect code issues during development as early as possible and to prevent these code issues to be included in the final product. Every test suite, independent from its ability to find code issues, is contributing to this goal. Unfortunately, executing test cases is expensive: every single test execution costs money and slows down product development. Test suites with a very low probability of finding code issues might be considered cost ineffective—test can only proof the presence of code issues but not their absence.

We use two different test suite effectiveness measurements:

Number of fixed bugs (*NumBugs*): The absolute number of distinct bug reports (also resolving duplicate and related bug reports) associated with any true test failure reported by the corresponding test suite. This measure is dependent on the execution frequency of the test suite.

Bug detection ratio per build (*BugsPerExec*): The relative number of *NumBugs* per test suite execution. This measure relates to the historic probability of a test suite to report at least one true test failure.

Although very similar, we explicitly used both, the absolute and the relative number of code issues detected by test suites. For the development process, the absolute number of code issues found is far more important than its relative correspondence. A test finding one fatal code errors might already prevent a disaster.

Note that we explicitly ignore common test quality measurements such as code coverage. The reason is that coverage does not state anything about the amount of executed lines actually checked for correctness. Every system and integration tests covers large parts of the Windows kernel and core Windows binaries, but only few of them specifically check for the correctness of these functionalities.

4.3 Test Suite Reliability

Similar to test suite effectiveness, we measure test suite reliability using the **relative number of test suite executions that reported at least one false test failure** (***FpPerExec***). Please note that a test suite might report more than one test failure per execution. Using this measurement, test suite reliability is measured in decimal numbers between zero and one. The value corresponds to the likelihood of a test suite to report a false test failure when executed. A value of zero means that the test suite never reported any false test failure in the past; a value of one indicates that all executions of the test suite reported at least one false test failure.

We did not account for differences between false test failures. One would imagine that a false test failure that triggers a bug report is worse than those that did not. However, this generalization is not valid. Most of the bug reports based on false test failures are documentation artifacts engineers can refer to in order to document known test issues.

While we defined two metrics for effectiveness, we only use one for reliability. The reason is that for effectiveness both, the absolute number of bugs as well as the relative number of bugs, can be relevant. This is not the case for reliability. While a single bug can have catastrophic consequences, a single false test failure does not.

Table 2: List of models used for classification experiments.

Model	Description
k-nearest neighbor (knn)	This model finds k training instances closest in Euclidean distance to the given test instance and predicts the class that is the majority amongst these training instances.
Logistic regression (multinorm)	This is a generalized linear model using a logic function and hence suited for binomial regression, i.e. where the outcome class is dichotomous.
Recursive partitioning (rpart)	A variant of decision trees, this model can be represented as a binomial tree and popularly used for classification tasks.
Support vector machine (svmRadial)	This model classifies data by determining a separator that distinguishes the data with the largest margin. We used the radial kernel for our experiments.
Tree bagging (treebag)	Another variant of decision trees, this model uses bootstrapping to stabilize the decision trees.
Random forest (randomForest)	An ensemble of decision tree classifiers. Random forests grow multiple decision trees each "voting" for the class on an instance to be classified.

5. EXPERIMENTAL SETUP

The goal of this paper is to investigate whether the organizational structure of test case owners impact the effectiveness or reliability of test suites. To that extend, we investigated the dependency between our organizational metrics (Section 0) and our test suite quality measurements (Section 4). More detailed, we measured the correlations between the individual measurements and investigated whether we can use organizational metrics to predict the effectiveness and reliability of the corresponding test suites. We conducted our investigation on Windows BVT test suites. We excluded all test suites that were executed in less than 10% of all official builds.

5.1 Correlations

To show basic relations between organizational structure and test suite effectiveness and reliability, we computed spearman rank correlations between the organizational metrics discussed in Section 3.1 and the test suite effectiveness (Section 4.2) and test suite reliability measures (Section 4.3). Correlation values lie between -1 and 1 and describes how well the dependency between two metrics can be described using a monotonic function. A correlation value of 1 or -1 occurs when one metrics is a perfect monotone function of the respectively other measurement. All reported metrics are statistically significant. We checked for significance using *cor.test* in R package, which uses Spearman's rho statistic to estimate a rank-based measure of association

5.2 Classification Models

Rank correlations are good indicators of whether a metric might be a good predictor for a dependent variable, it does not allow to draw precise conclusions on how well a predictor that combines multiple measurements will be. To investigate how well metrics capturing organizational structure can be used to predict effectiveness and reliability of test suites we used actual classification models.

Table 3: Correlations between organizational and test suite effectiveness measures.

Metric	NumFixedBugs	BugsPerExec
LongestDevDistance	0.67	-0.23
MedianDevDistance	0.64	-0.38
MeanDevDistance	0.68	-0.28
MeanPathLength	0.73	-0.25
SizeOfLargestClique	0.23	0.24
NumOwner	0.87	-0.17
NumL2	0.61	-0.26
NumL3	0.70	-0.14
NumL4	0.69	-0.17
NumL5	0.68	-0.08
NumOwnerLeftOrg	0.59	-0.30
PCOwnerLeftOrg	0.56	-0.14
HighestCommonManager	-0.64	0.16
NumAliasGroups	0.84	-0.34
MedianAliasGroupSize	0.37	0.32
NumSystemAlias	0.75	-0.38

In order to perform the experiments on a single Windows release, we had to sample the dataset into two subsets—training and testing sets. The training set is used to train the classification model that we evaluate on the corresponding testing set. To split the overall dataset into these two subsets, we used a *stratified repeated holdout setup*—the data is stratified 9before sampling) to preserve the proportion of positive and negative instances in the data in both training and testing sets. We sampled the data 100 times and used two third of the sampled data for training and the remaining one third for testing purposes. We report the mean precision and recall values aggregating the individual prediction results.

To test if prediction models are dependent upon machine leaning algorithms, we used six different prediction algorithms further described in Table 2. Each of these models is performed on exactly the same cross-folds to allow fair comparison. For a more detailed description of the used models, we advise the reader to refer to specialized machine learning texts such as by Witten and Frank [7].

We conducted our experiments using R-statistical software [8] and Max Kuhn's R package *caret* [9].

Predicting Test Suite Effectiveness

Regarding test suite effectiveness, we trained and tested classification models to classify test suites being above-median effective—test suites whose test suite effectiveness measure exceeds the median value of all test suite effectiveness measures. We used two different dependent variables as effectiveness indicator: *NumBugs* and *BugsPerExec* (see Section 4.2). For example, if test suite T reported more than the median number of fixed bugs for all test suites, we would classify T as above-median efficient and expect our model to predict T to be in that category.

Predicting Test Suite Reliability

Regarding test suite reliability, we trained and tested models to classify test suites to report an above-median number of false failures per build: *FpPerExec* (see Section 4.3).

Table 4: Classification accuracy for models predicting test suites associated with above-median *NumFixedBugs* (left part) and above-median *BugsPerExec* (right part).

Model	*NumFixedBugs*		*BugsPerExec*	
	Precision	Recall	Precision	Recall
multinom	0.88	0.87	0.69	0.64
nb	0.92	0.71	0.70	0.52
rf	0.85	0.87	0.70	0.66
rpart	0.83	0.99	0.61	0.72
svmRadial	0.84	0.97	0.66	0.70
treebag	0.85	0.84	0.70	0.67

5.3 Metric Importance

To estimate the metric importance of each organizational metric for the corresponding classification model, we used the *filterVarImpl* function of the caret package [9] to conduct a series of ROC curve analysis for each metric: "a series of cutoffs is applied to the predictor data to predict the class. The sensitivity and specificity are computed for each cutoff and the ROC curve is computed. The trapezoidal rule is used to compute the area under the ROC curve. This area is used as the measure of variable importance." [4].
Please note that the metric importance for classification models may not match the spearman rank correlation results. While the rank correlation considers the exact order of entities, classification models separate entities into two categories. The suitability of a metric to solve either problem may be different.

6. RESULTS

In this section, we discuss the results of all experimental setups described in Section 5.

6.1 Organizational Structure and Test Effectiveness

Metrics correlations

The correlations between organizational measurements and metrics expressing the effectiveness of test suites are shown in Table 3.

The relative number of bugs reported by a test suite per execution (column three in Table 3) is negatively correlated with organizational path and higher level manager metrics (e.g. *NumL2*). This suggests that test suites with owners stemming from the same organizational subgroups are more effective with respect to finding code issues. The positive correlation between *BugsPerExec* and *SizeOfLargestClique* further supports this trend.

> ⚐ Test Suites owned by a larger organizational subgroups with short communication paths tend to be more effective.

Interesting is also the relationship between the absolute and relative number of test owners that left the company (*NumOwnerLeftOrg* and *PCOwnerLeftOrg*) and test effectiveness (*BugsPerExec*). The correlation values might be weak, but it still indicates that tests owned by people that left the organization seem to be less valuable that test cases owned by current employees.

> ⚐ Test Suites owned by engineers that left the company seem to be less effective.

The number of email aliases assigned as test owners per test suite is negatively correlated with *BugsPerExec* but strongly, positively correlated with *NumFixedBugs*. There exist two possible interpretations for this. Either does an email group as test owner

Table 6: Metric importance for models predicting *NumFixedBugs* (see Table 4) ordered by importance.

Metric	Area under ROC
NumOwner	0.80
NumAliasGroups	0.77
NumL5	0.76
LongestDevDistance	0.76
NumL4	0.74
NumL3	0.74
MeanPathLength	0.73
MeanDevDistance	0.71
HighestCommonManager	0.71
NumL2	0.67
MedianDevDistance	0.63
NumOwnerLeftOrg	0.63
MedianAliasGroupSize	0.60

Table 5: Metric importance for models predicting *BugsPerExec* (Table 4) ordered by importance.

Metric	Area under ROC
MedianDevDistance	0.78
MeanDevDistance	0.72
NumL2	0.69
MeanPathLength	0.69
NumOwnerLeftOrg	0.68
NumAliasGroups	0.68
LongestDevDistance	0.68
MedianAliasGroupSize	0.67
NumL4	0.65
HighestCommonManager	0.64
NumOwner	0.63
NumL3	0.63
NumL5	0.59

suggest no clear ownership, or it might be that an email alias as test owner suggests that these test steps are more general test steps (such as setup and tear down steps) that by nature are more likely to find general code issues and get executed more frequently—therefore the strong correlation with the absolute number of bugs. Both these results, i.e. test suites owned by engineers who have left the company and test suite tasks with a high number of email aliases show the importance of test ownerships, the results of which are analogous to our prior code ownership results [5].

> ⚐ Test suite tasks with a higher number of email aliases as owners show negative correlations with the relative number of bugs per execution. The size of email aliases is positively correlated.

Table 7: Correlations of organizational metrics and the relative number of false failures (*FpPerExec*).

Metric	FpPerExec
LongestDevDistance	0.39
MedianDevDistance	0.33
MeanDevDistance	0.34
MeanPathLength	0.39
SizeOfLargestClique	0.14
NumOwnerLeftOrg	0.27
NumL2	0.32
NumL3	0.36
NumL4	0.36
NumL5	0.35
NumOnwer	0.09
PCOwnerLeftOrg	0.08
HighestCommonManager	-0.36
NumAliasGroups	0.19
MedianAliasGroupSize	0.11
NumSystemAlias	0.19

Classification Accuracy

Classification results for the absolute number of fixed bugs are shown in the left part of Table 4. Precision and recall values for the absolute number of fixed bugs are high: precision between 0.8 and 0.9, recall values between 0.7 and 1.0.

Results for models classifying test suites haven an above-median relative number of code issues detected is shown in right part of Table 4 and show precision values around 0.69 (median precision) and recall values around 0.66 (median recall). Although the prediction accuracy is moderate, organizational structure seems to impact the ability of tests to find and report code issues. The results also show that the prediction accuracies across different machine learning models is similar and show no significant difference.

> ⚐ Using test suite metrics, we are able to build prediction models that predict the effectiveness of test suites with precision and recall values around 0.7.

Metrics Importance

The metric importance for our classification models are shown in Table 6 and Table 5. As expected, the number of owners dominate the classification models predicting absolute number of detected code issues (Table 6). Models predicting relative number of detected code issues (Table 5) are dominated by developer distance and the number of high level managers. From the correlation values discussed above we know that developer distance metrics are negatively correlated with the relative number of code issues.

6.2 Organizational Structure and Test Reliability

In this section, we discuss the results of our experiments investigating the dependency between organizational metrics and test suite reliability.

Table 9: Classification results for models predicting the relative number of false test suite failures (*FpPerExec*).

Model	Precision	Recall
multinom	0.93	0.89
nb	0.93	0.76
rf	0.95	0.87
rpart	0.84	0.91
svmRadial	0.86	0.91
treebag	0.93	0.88

Metrics correlations

The correlations between organizational metrics and the relative number of false failures are shown in Table 7. Although *FpPerExec* is a relative number, we do not observe any strong correlations. While the correlations between test suite effectiveness and communication path lengths was negative, the correlations between these path lengths and our reliability measurement are positive. Thus, while indicative that the longer the communication paths, the more the false failures faced we do not draw an inferences due to the absence of really strong correlations as in earlier results.

Classification Accuracy

Table 9 shows that organizational metrics can be excellent predictors for test suite reliability. With precision values between 0.84 and 0.93 (median precision across all machine learning algorithms at 0.93), only 7% of all classified test suites predicted to be less reliable than median are wrongly classified as such. Recall values are high as well and lie between 0.76 and 0.9. This result is surprising as it indicates that the vast majority of test suite reliability issues can be explained by organizational metrics.

> Organizational structure metrics are excellent predictors for test suite reliability issues and should be considered as strong indicators.

Metrics Importance

Table 8 contains metric importance measurements for our reliability prediction models and shows that the number of owners and team email aliases as well as the developer distances are most important. As discussed previously, the length of the shortest paths seems to be the most critical metrics. Short shortest paths indicate high effectiveness, long shortest paths indicate low reliability.

7. RELATED WORK

A number of prior studies investigates the impact of organizational structure and code ownership on software quality. There also exists prior work describing driving factors for test effectiveness and reliability and how to increase test quality. But to the best of our knowledge, there have been no studies connecting organizational structure with test quality.

Ownership and Organizational Structure

Weyuker et al. [10] studies the effect of development team size on code quality. The authors used the number of engineers contributing to code artifacts as quality indicator. Similar, Meneely and William [11] related the number of engineers contributing to code entities to security vulnerabilities. Later, Rahman and Devanby [3] used extended team size measures capturing the organizational structure of contributors to investigated the impact of code ownership and engineer experience on code quality. In fact, domain knowledge has shown to be an important reason for

Table 8: Metric importance for models predicting the relative number of false test suite failures (*FpPerExec*).

Metric	Area under ROC
NumOwner	0.96
NumAliasGroups	0.91
MeanPathLength	0.89
NumL4	0.89
NumL3	0.88
LongestDevDistance	0.88
MeanDevDistance	0.87
HighestCommonManager	0.86
NumL2	0.83
NumL5	0.82
MedianDevDistance	0.81
NumOwnerLeftOrg	0.75
MedianAliasGroupSize	0.66

software issues [12]. Robillard [13] showed that the lack of domain knowledge negatively affects the quality of software and Mockus and Weiss [14] found that changes made by more experienced developers were less likely to induce code issues. Mockus also showed that "recent departures from an organization were associated with increased probability of customer-reported defects" [15] while Karus and Dumas [16] showed that organizational metrics can also be used to estimate yearly cumulative code churn.

Bird et al. [4,17] and Nagappan et al. [5] extended earlier studies by extending the definition of code ownership modelling the actual proportion of work individual engineers contributed to a software artifact. In their studies on Microsoft Windows, the authors established a "statistical significant relationship between ownership and failures" [4] that can be used to build reliable defect prediction models. It seems important that "managers need to be able to assess the communication patterns and deficiencies that exist in a development team and support the establishment of communication paths that are structured in a particular way to help the team's outcomes." [18]. To raise awareness of organizational structures and possible software quality implications, Basili and Caldiera [19] presented an approach to improve software quality through learning and experience by establishing "experience factories". Similar, Tamburri et al. provided "instruments allowing practitioners to identify, select, analyze, or support the exact social structure they need" [20]. Lately, Bettenburg and Hassan developed "statistical models to study the impact of social interactions in a software project on software quality" [21]. Instead of using metrics on rather static organization structure information, the authors used social information mined from the issue tracking and version control repositories of two large open-source software projects. Their results show that social interaction metrics complement traditional code metrics used for defect prediction purposes.

All of these previous studies concentrate on the dependency between organizational, social metrics and code quality, but do not explore the effect of these metrics on software testing quality. Although test quality and product quality might be closely related, the study does not all a direct connection between organizational structure and test behavior. The study presented in this paper differs

in that respect as we studies explicitly the effect of organizational structure on software test effectiveness and reliability.

Test Effectiveness and Test Reliability

As Basili mentioned: "Measuring the absolute effectiveness of testing is generally not possible, but comparison between effectiveness of tests is" [22]. Keeping this in mind, most (empirical) studies on test effectiveness show comparisons between individual test strategies.

Basili and Selby [23] presented one of the earliest studies comparing the effectiveness and cost of software testing strategies showing that changing or choosing different test strategies might impact the effectiveness of testing processes. Consequently, many test selection and prioritization efforts use fault detection measures as test selection criteria [24,25]. A number of empirical studies and extensive literature reviews compare and identify test tools most likely to yield optimal test effectiveness [26,27,28,29].

In the information storage and retrieval domain, test reliability measures seem to be based on the number and quality of queries a test suite contains [30,31,32], rather than on the actual number of code issues detected by these queries.

To the best of our knowledge there has been no study on test reliability by measuring the number of test failures caused by other test and infrastructure issues. Most studies consider unit tests and measure the effectiveness and reliability based on their ability to fail due to code issues. However, test failures due to other reasons than code issues are likely to impact product development and thus should be considered harmful and taken into account when measuring test quality.

8. THREATS TO VALIDITY

Like most empirical studies, the presented study has threats to validity. We identified two main groups of threats.

8.1 Generalizability

In this study we investigated quality test suites specific to the Windows development process. Even though the individual test processes might be Windows specific, the (continuous) execution of test suites during software development and measurements for effectiveness and reliability are not.

As discussed, Windows build verification test suites contain test cases contributed by engineers working for different development teams. Test suites whose test content is entirely owned by the test team itself are by nature more uniform in terms of organizational structure and test content. Replicating or extending the study presented in this paper to other test suites or even other products and projects might lead to different results and conclusions.

8.2 Construct Validity

Our study relies on the correctness of the organizational tree and the correctness of the associations between test cases and test owners. The organizational tree used to compute our organizational metrics is using datasets provided by the CODEMINE [6] project. Any issues of CODEMINE regarding the organizational structure might also affect the presented results. To identify test case owners, we used official database provided by the BVT testing team, but we were unable to verify the correctness of test owners. Any incorrect entry in this dataset might impact the results presented in this paper.

Ideally the effectiveness of test should be measured as the ratio between the number of defects found by the test and the number of defects inside the tested code area. However, this would require to know the total number of defects in Windows code. Since this number is unknown, we can only operate on the known number of defects rather than the unknown total number of defects.

9. SUMMARY AND IMPLICATIONS

In this study, we analyze the impact of organization structure on test suite effectiveness and reliability. Using organizational metrics, we are able to build prediction models to

- Identify test suites with above-median effectiveness with precision and recall values around 0.7.
- Identify test suites with below-median reliability (number of false failures) with precision values around 0.8 and recall values around 0.9.

Thus, organization structure seems to explain large parts of test suite effectiveness and reliability issues. Metric importance and correlation values suggest that test suites whose owners are distributed over multiple organization subgroups with long communication paths are negatively correlated with quality. As a consequence, we suggest to review test suites with respect to their organizational composition and to support test suites that are clearly owned by individual organizational subgroups. As indicated by the results of this study, this would increase the effectiveness and reliability of test suites.

Increasing effectiveness and reliability of test suites is also likely to improve development productivity. Test failures can slow down productivity, require human triaging, and block code integration until the test failure is either resolved (test now passing) or until the failure is verified to be a false failure. These development activities are expensive in terms of cost and time as they involve human interaction. Increasing test reliability and effectiveness is likely to increase development speed and to improve developer satisfaction.

The results of this study are aligned with results of previous Microsoft studies investigating the effect of code ownership on code quality [1] showing that distributed code ownership can have severe impact on code quality: the higher the number of engineers contributing to a source code entity (e.g. binary or file) the higher the lower the code quality of that code entity. These results on production code and our results on test suites are surprisingly aligned and show the same trend: code and test ownership are important properties and should be considered as a driving factor for code and test issues.

10. ACKNOWLEDGMENTS

We thank the Windows development and BVT quality team for their tremendous support and feedback. This work is based on data extracted from varies development repositories provided by the Microsoft TSE group. Our special thanks go to Blerim Kuliqi, Jason Means, and Poornima Priyadarshini.

11. REFERENCES

[1] Bird, C., Nagappan, N., Murphy, B., Gall, H., and Devanbu, P. Don'T Touch My Code!: Examining the Effects of Ownership on Software Quality. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering* (2011), ACM, 4--14.

[2] Nordberg, M.E., III. Managing code ownership. *Software, IEEE*, 20 (Mar 2003), 26-33.

[3] Rahman, F. and Devanbu, P. Ownership, Experience and Defects: A Fine-grained Study of Authorship. In *Proceedings of the 33rd International Conference on Software Engineering* (2011), ACM, 491--500.

[4] Bird, C. and Zimmermann, T. Assessing the Value of Branches with What-if Analysis. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering* (Cary, North Carolina, 2012), ACM, 45:1--45:11.

[5] Nagappan, N., Murphy, B., and Basili, V. The Influence of Organizational Structure on Software Quality: An Empirical Case Study. In *Proceedings of the 30th International Conference on Software Engineering* (2008), ACM, 521--530.

[6] Czerwonka, J., Nagappan, N., Schulte, W., and Murphy, B. CODEMINE: Building a Software Development Data Analytics Platform at Microsoft. *Software, IEEE*, 30, 4 (2013), 64--71.

[7] Witten, I.H. and Frank, E. Data mining: practical machine learning tools and techniques with Java implementations. *SIGMOD Rec.*, 31 (mar 2002), 76--77.

[8] Team, R.D.C. *R: A Language and Environment for Statistical Computing*. , 2010. R Foundation for Statistical Computing.

[9] Kuhn, M. *caret: Classification and Regression Training*. , 2011.

[10] Weyuker, E.J., Ostrand, T.J., and Bell, R.M. Do too many cooks spoil the broth? using the number of developers to enhance defect prediction models. *Empirical Software Engineering*, 13 (2008), 539--559.

[11] Meneely, A. and Williams, L. Secure Open Source Collaboration: An Empirical Study of Linus' Law. In *Proceedings of the 16th ACM Conference on Computer and Communications Security* (2009), ACM, 453--462.

[12] Curtis, B., Krasner, H., and Iscoe, N. A Field Study of the Software Design Process for Large Systems. *Commun. ACM*, 31 (nov 1988), 1268--1287.

[13] Robillard, P.N. The Role of Knowledge in Software Development. *Commun. ACM*, 42 (jan 1999), 87--92.

[14] Mockus, A. and Weiss, D.M. Predicting risk of software changes. *Bell Labs Technical Journal*, 5 (2000), 169--180.

[15] Mockus, A. Organizational Volatility and Its Effects on Software Defects. In *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering* (2010), ACM, 117--126.

[16] Karus, S. and Dumas, M. Code Churn Estimation Using Organisational and Code Metrics: An Experimental Comparison. *Inf. Softw. Technol.*, 54 (feb 2012), 203--211.

[17] Bird, C., Nagappan, N., Devanbu, P., Gall, H., and Murphy, B. Does Distributed Development Affect Software Quality? An Empirical Case Study of Windows Vista. In *Proceedings of the 31st International Conference on Software Engineering* (2009), IEEE Computer Society, 518--528.

[18] Cataldo, M. and Ehrlich, K. The Impact of the Structure of Communication Patterns in Global Software Development: An Empirical Analysis of a Project Using Agile Methods. *Institute for Software Research, Carnegie Mellon University* (2011).

[19] Basili, V.R. and Caldiera, G. Improve Soft-ware Quality by Reusing Knowledge and Experience. *Sloan management review*, 37 (1995), 55--55.

[20] Tamburri, D.A., Lago, P., and Vliet, H.v. Organizational Social Structures for Software Engineering. *ACM Comput. Surv.*, 46 (jul 2013), 3:1--3:35.

[21] Bettenburg, N. and Hassan, A.E. Studying the Impact of Social Interactions on Software Quality. *Empirical Softw. Engg.*, 18 (apr 2013), 375--431.

[22] Weyuker, E.J. Can we measure software testing effectiveness? In *Software Metrics Symposium, 1993. Proceedings., First International* (May 1993), 100-107.

[23] Basili, V.R. and Selby, R.W. Comparing the Effectiveness of Software Testing Strategies. *IEEE Trans. Softw. Eng.*, 13, 12 (December 1987), 1278--1296.

[24] Goradia, T. Dynamic impact analysis: a cost-effective technique to enforce error-propagation. In *Proceedings of the 1993 ACM SIGSOFT international symposium on Software testing and analysis* (Cambridge, Massachusetts, USA, 1993), ACM, 171--181.

[25] Zhang, Y., Zhao, X., Zhang, X., and Zhang, T. Test effectiveness index: Integrating product metrics with process metrics. In *Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2012 IEEE International Conference on* (May 2012), 54-57.

[26] Whyte, G. and Mulder, D.L. Mitigating the Impact of Software Test Constraints on Software Testing Effectiveness. *Electronic Journal of Information Systems Evaluation*, 14 (2011), 254 - 270.

[27] Rothermel, G., Untch, R.H., Chu, C., and Harrold, M.J. Test case prioritization: an empirical study. In *Software Maintenance, 1999. (ICSM '99) Proceedings. IEEE International Conference on* (1999), 179-188.

[28] Elbaum, S., Malishevsky, A.G., and Rothermel, G. Test case prioritization: a family of empirical studies. *Software Engineering, IEEE Transactions on*, 28 (Feb 2002), 159-182.

[29] Braione, P., Denaro, G., Mattavelli, A., Vivanti, M., and Muhammad, A. An industrial case study of the effectiveness of test generators. In *Automation of Software Test (AST), 2012 7th International Workshop on* (June 2012), 50-56.

[30] Brennan, R.L. Generalizability theory. *Educational Measurement: Issues and Practice*, 11 (1992), 27--34.

[31] Shavelson, R.J. and Webb, N.M. *Generalizability theory: A primer*. Sage, 1991.

[32] Urbano, J., Marrero, M., and Mart\'\in, D. On the Measurement of Test Collection Reliability. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval* (2013), ACM, 393--402.