The Influence of Organizational Structure on Software Quality: An Empirical Case Study

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

Brendan Murphy
Microsoft Research
Cambridge, UK
bmurphy@microsoft.com

Victor R. Basili
University of Maryland
College Park, MD, USA
basili@cs.umd.edu

ABSTRACT
Often software systems are developed by organizations consisting of many teams of individuals working together. Brooks states in the *Mythical Man Month* book that product quality is strongly affected by organization structure. Unfortunately there has been little empirical evidence to date to substantiate this assertion. In this paper we present a metric scheme to quantify organizational complexity, in relation to the product development process to identify if the metrics impact failure-proneness. In our case study, the organizational metrics when applied to data from Windows Vista were statistically significant predictors of failure-proneness. The precision and recall measures for identifying failure-prone binaries, using the organizational metrics, was significantly higher than using traditional metrics like churn, complexity, coverage, dependencies, and pre-release bug measures that have been used to date to predict failure-proneness. Our results provide empirical evidence that the organizational metrics are related to, and are effective predictors of failure-proneness.

Categories and Subject Descriptors
D.2.8 [**Software Engineering**]: Software Metrics – *complexity measures, performance measures, process metrics, product metrics.*

General Terms
Measurement, Reliability, Human Factors.

Keywords
Organizational structure, Failures, Code churn, Developers, Software mining, Empirical studies.

ICSE'08, May 10–18, 2008, Leipzig, Germany.

1. INTRODUCTION
Software engineering is a complex engineering activity. It involves interactions between people, processes, and tools to develop a complete product. In practice, commercial software development is performed by teams consisting of a number of individuals ranging from the tens to the thousands. Often these people work via an organizational structure reporting to a manager or set of managers.

The intersection of people [9], processes [29] and organization [33] and the area of identifying problem prone components early in the development process using software metrics (e.g. [12, 23, 27, 30]) has been studied extensively in recent years. Early indicators of software quality are beneficial for software engineers and managers in determining the reliability of the system, estimating and prioritizing work items, focusing on areas that require more testing, inspections and in general identifying "problem-spots" to manage for unanticipated situations. Often such estimates are obtained from measures like code churn, code complexity, code coverage, code dependencies, etc. But these studies often ignore one of the most influential factors in software development, specifically "people and organizational structure". This interesting fact serves as our main motivation to understand the intersection between organizational structure and software quality: *How does organizational complexity influence quality? Can we identify measures of the organizational structure? How well do they do at predicting quality, e.g., do they do a better job of identifying problem components than earlier used metrics?*

Conway's Law states that "organizations that design systems are constrained to produce systems which are copies of the communication structures of these organizations." [8]. Similarly, Fred Brooks argues in the Mythical Man Month [6] that the product quality is strongly affected by org structure. With the advent of global software development where teams are distributed across the world the impact of organization structure on Conway's law [14] and its implications on quality is significant. To the best of our knowledge there has been little or no empirical evidence regarding the relationship/association between organizational structure and direct measures of software quality like failures.

In this paper we investigate this relationship between organizational structure and software quality by proposing a set of eight measures that quantify organizational complexity. These eight measures provide a balanced view of organizational complexity from the code viewpoint. For the organizational metrics, we try to capture issues such as organizational distance of the developers; the number of developers working on a

component; the amount of multi-tasking developers are doing across organizations; and the amount of change to a component within the context of that organization etc. from a quantifiable perspective. Using these measures we empirically evaluate the efficacy of the organizational metrics to identify failure-prone binaries in Windows Vista.

The organization of the rest of the paper is as follows. Section 2 describes the related work focusing on prior work on organizational structure and predicting defects/failures. Section 3 highlights our contribution and Section 4 describes the organizational metric suite. Section 5 presents our case study and the results of our investigation on the relationship between organizational metrics and quality. Section 6 discusses the threats to validity and section 7 the conclusions and future work.

2. RELATED WORK

Our discussion of related work falls into one of the following two categories: Organizational research from the software perspective and predicting faults/failures.

2.1 Software Organizational Studies

From the historical perspective, Fred Brooks in his classic book *The Mythical Man Month* [6] provides an analogy in the chapter on *Why did the (mythical) Tower of Babel Fail?* The observation being that, the people had (1) a clear mission; (2) manpower; (3) (raw) materials; (4) time and (5) technology. The project failed because of – *communication, and its consequent organization* [6]. Brooks further states that in software systems: schedule disasters, functional misfits and system bugs arise from a lack of communication between different teams. Quoting Brooks[6] "T*he purpose of organization is to reduce the amount of communication and coordination necessary; hence organization is a radical attack on the communication problems…".* In 1968 Conway [8] also observed from his study (organizations produce designs which are copies of the communication structures of these organizations) that the flexibility of an organization is important to effective design [8]. He further went on to say that ways must be found to reward design managers for keeping their organizations lean and flexible indicating the importance of organization on design quality [8]. In a similar vein, Parnas [32] also indicated that a software module is "a responsibility assignment rather than a subprogram" indicating the importance of organizational structure in the software industry.

We summarize here recent work from the perspective of organizational structure towards communication and coordination. Herbsleb and Grinter [13] look at Conway's law from the perspective of global software development. Their paper explores global software development from a team organizational context based on teams working in Germany and UK. They provide recommendations based on their empirical case study for the associated problems geographically distributed organizations face with respect to communication barriers and coordination mechanisms. They observed the primary barriers to team coordination were lack of unplanned contact; knowing the right person to contact about specific issues; cost of initiating the contact; effective communication and lack of trust. Further Herbsleb and Mockus [15] formulate and evaluate an empirical theory (of coordination) towards understanding engineering decisions from the viewpoint of coordination within software projects. This paper is one of the closest in scale, size and motivation to our study, though our study focuses on predicting quality using the organization metrics (with the underlying relationship between organizational structure and coordination). Also Mockus et al. [22] investigate how different individuals across geographical boundaries contribute towards open source projects (Apache and Mozilla). Perry et al. [33] discuss and motivate the need to consider the larger development picture, which encompasses organizational and social as well as technological factors. They discuss quantitatively measuring people factors and report on the result of two experiments, one which is a self-reported diary of developer activities and the second an observational study of developer activities. These two experiments also were used to asses the efficacy of each technique towards quantifying people factors.

2.2 Software Metrics and Faults/Failures

In this section we summarize some of the related work regarding metrics and faults/failures. Relevant studies on Microsoft systems are also presented providing context and for comparison to our current work. We organize our work based on the type of metrics that have been studied for fault/failures prediction.

Code Churn: Graves et al. [12] predict fault incidences using software change history based on a weighted time damp model using the sum of contributions from all changes to a module, where large and/or recent changes contribute the most to fault potential [12]. Ostrand et al. [31] use information of file status such as new, changed, unchanged files along with other explanatory variables such as lines of code, age, prior faults etc. as predictors in a negative binomial regression equation to successfully predict (high accuracy for faults found in both early and later stages of development) the number of faults in a multiple release software system. Nagappan and Ball [25] in a prior study on Windows Server 2003 showed the use of relative code churn measures (relative churn measures are normalized values of the various measures obtained during the evolution of the system) to predict defect density at strong statistically significant levels. Zimmermann et al. [37] mined source code repositories of eight large scale open source systems (IBM Eclipse, Postgres, KOffice, gcc, Gimp, JBoss, JEdit and Python) to predict where future changes will take place in these systems. The top three recommendations made by their system identified a correct location for future change with an accuracy of 70%.

Code Complexity: Khoshgoftaar et al. [18] studied two consecutive releases of a large legacy system (containing over 38,000 procedures in 171 modules) for telecommunications. Discriminant analysis identified fault-prone modules based on 16 static software product metrics. Their model when used on the second release showed a type I and II misclassification rate of 21.7%, 19.1% respectively and an overall misclassification rate of 21.0%. From the O-O (object-oriented) perspective the CK metric suite [7] consist of six metrics (designed primarily as object oriented design measures): weighted methods per class (WMC), coupling between objects (CBO), depth of inheritance (DIT), number of children (NOC), response for a class (RFC) and lack of cohesion among methods (LCOM). The CK metrics have also been investigated in the context of fault-proneness. Basili et al. [1] studied the fault-proneness in software programs using eight student projects. They observed that the WMC, CBO, DIT, NOC and RFC were correlated with defects while the LCOM was not correlated with defects. Further, Briand et al. [5] performed an

industrial case study and observed the CBO, RFC, and LCOM to be associated with the fault-proneness of a class. Within five Microsoft projects, Nagappan et al. [27] identified complexity metrics that predict post-release failures and reported how to systematically build predictors for post-release failures from history.

Code Dependencies: Pogdurski and Clarke [34] presented a formal model of program dependencies as the relationship between two pieces of code inferred from the program text. Schröter et al. [35] showed that import dependencies can predict defects. They proposed an alternate way of predicting failures for Java classes. Rather than looking at the complexity of a class, they looked exclusively at the components that a class uses. For Eclipse, the open source IDE they found that using compiler packages results in a significantly higher failure-proneness (71%) than using GUI packages (14%). Prior work at Microsoft [24] on the Windows Server 2003 system illustrates that code dependencies can be used to successfully identify failure-prone binaries with precision and recall values of around 73% and 75% respectively.

Code Coverage: Hutchins et al. [16] evaluate all-edges and all-uses coverage criteria using an experiment with 130 fault seeded versions of seven programs and observed that test sets achieving coverage levels over 90% usually showed significantly better fault detection than randomly chosen test sets of the same size. In addition, significant improvements in the effectiveness of coverage-based tests usually occurred as coverage increased from 90% to 100%. Frankl and Weiss [11] evaluated all-edges and all-uses coverage using nine subject programs. Error-exposing ability was shown to be positive and strongly correlated to percentage of covered definition-use associations in four of the nine subjects. Error exposing ability was also shown to be positively correlated with the percentage of covered edges in four (different) subjects, but the relationship was weaker.

Combination of metrics: Denaro et al. [10] calculated 38 different software metrics (lines of code, halstead software metrics, nesting levels, cyclomatic complexity, knots, number of comparison operators, loops etc.) for the open source Apache 1.3 and Apache 2.0 projects. Using logistic regression models built using the data collected from the Apache 1.3 they verified the models against the Apache 2.0 project with high correctness/completeness. Khoshgoftaar et al. [19] use code churn as a measure of software quality in a program of 225,000 lines of assembly language. Using eight complexity measures, including code churn, they found neural networks and multiple regression to be an efficient predictor of software quality, as measured by gross change in the code. Nagappan et al. [26] used code churn, code complexity and code coverage measures to predict post-release field failures in Windows Server 2003 using logistic regression models built with Windows XP data. The built models identify failure-prone binaries with a statistically significant positive and strong correlation between actual and estimated failures.

Pre-release bugs: Biyani and Santhanam [4] show for four industrial systems at IBM there is a very strong relationship between development defects per module and field defects per module. This allows building of prediction models based on development defects to identify field defects.

3. CONTRIBUTIONS

Our work extends the state of the art in the following ways.

1. The introduction, definition and use of an organizational metric suite specifically targeted at the software domain.
2. A methodology to systematically build predictors for failure-proneness using organizational structure metrics.
3. An investigation of whether organizational metrics are better predictors of failure-proneness compared to traditional code churn, code complexity, code dependencies, code coverage and pre-release defects.
4. It quantifies institutional knowledge in terms of developer experience on prior versions of Windows to define a baseline for other systems and applications outside of Microsoft.
5. It is one of the largest studies of commercial software—in terms of code size (> 50 Million lines of code), team sizes (several thousand), and software users (several Million).

4. ORGANIZATIONAL METRICS

In this section we will explain the organizational metrics that were developed for the purpose of our study. These metrics and their interactions were refined using the G-Q-M (Goal-Question-Metric) approach [2]. To explain the measures better we use a pseudo example shown in Figure 1 to represent the organizational structure of a company "XYZ".

Context: As a background to our example consider the measurement of the organizational metrics for a binary A.dll developed by company "XYZ". Over the course of its development prior to its release, the total number of edits for the files that were compiled into A.dll is 250. In Figure 1, Person A is the overall head of the company and manages the 100 person organization. Person AB manages a 30 person organization, AC manages a 40 person organization, AD manages a 30 person organization representing the three organizations within the company. The rest of the sub-managers, frontline engineers are also shown in Figure 1. We now define the eight organizational measures to quantify the organization complexity of company "XYZ" from the perspective of software development: in our case binary A.dll.

1. Number of Engineers (NOE): This is the absolute number of unique engineers who have touched a binary and are still employed by the company.
Implication*:* The more people who touch the code, the higher the chances of defective code as there is a higher need for coordination amongst the engineers[6]. Brooks [6] states that if there are N engineers who touch a piece of code there needs to be (N*(N-1))/2 theoretical communication paths for the N engineers to communicate amongst themselves. In our case if there is a large number of engineers who work on a particular binary there may be miscommunication between those engineers leading to design mismatches, breaking another engineers code (build breaks), and problem understanding design rationale.
Example: In this example this is a straight forward measurement of 32 engineers extracted from the version control system (VCS).

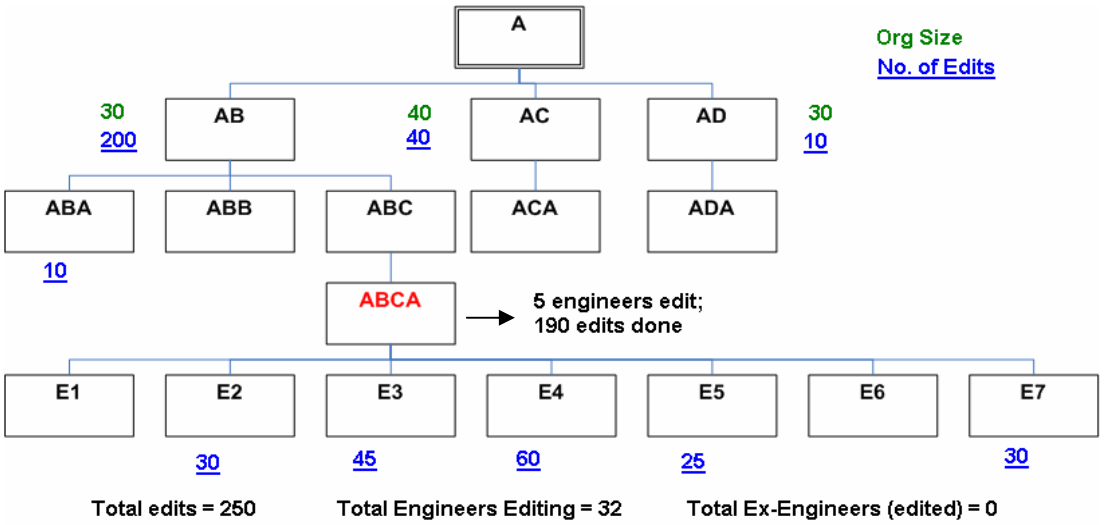

Figure 1: Example Organization Structure of Company "XYZ"

2. Number of Ex-Engineers (NOEE): This is the total number of unique engineers who have touched a binary and have left the company as of the release date of the software system (in our case A.dll).
Implications: This measure deals with knowledge transfer. If the employee(s) who worked on a piece of code leaves the company then there is a likelihood that the new person taking over might not be familiar with the design rationale, the reasoning behind certain bug fixes, and information about other stake holders in the code.
Example: This measure too is a straight forward value extracted from the VCS and checking against the org structure. In this example there were zero ex-engineers.

3. Edit Frequency (EF): This is the total number times the source code, that makes up the binary, was edited. An edit is when an engineer checks code out of the VCS, alters it and checks it back in again. This is independent of the number of lines of code altered during the edit.
Implications: This measure serves two purposes. One being that, if a binary had too many edits it could be an indicator of the lack of stability/control in the code from the different perspectives of reliability, performance etc. , this is even if a small number of engineers where making the majority of the edits. Secondly, it provides a more complete view of the distribution of the edits: did a single engineer make majority of the edits, or were they widely distributed amongst the engineers?. The EF cross balances with NOE and NOEE to make sure that a few engineers making all the edits do not inflate our measurements and ultimately affect our predict model. Also if the engineers who made most of the edits have left the company (NOEE) then it can lead to the above discussed issues of knowledge transfer.
Example: In our example the edit frequency is 250 also extracted from the VCS.

4. Depth of Master Ownership (DMO): This metric determines the level of ownership of the binary depending on the number of edits done. The organization level of the person whose reporting engineers perform more than 75% of the rolled up edits is deemed as the DMO. The DMO metric determines the binary owner based on activity on that binary. Our choice of 75% is based on prior historical information on Windows to quantify ownership.
Implications: The deeper in the tree is the ownership the more focused the activities, communication, and responsibility. A deeper level of ownership indicates less diffusion of activities, a single point of approval/control which should improve intellectual control. If a binary does not have a clear owner (or has a very low DMO at which 75% of the edits toll up) then there could be issues regarding decision-making when performing a risky bug fix, lack of engineers to follow-up if there is an issue, understanding intersecting code dependencies etc. A management owner who has not made a large number of edits (i.e. not familiar with the code) may not be able to make the above decisions without affecting code quality.
Example: In our above example more than 75% of the edits roll up to the engineer ABCA (190 edits out of a total of 250). Hence the DMO measure in this case is 2 (level 0 is AB, AC and AD; Level 1 is ABA to ADA. Person A being the top person is not involved in the technical day to day activities). The overall org owner for this org is AB.

5. Percentage of Org contributing to development (PO): The ratio of the number of people reporting at the DMO level owner relative to the Master owner org size.
Implications: The lower the percentage the more local is the ownership and contributions to the binary leading to lower coordination/communication overhead across organizations and improved synchronization amongst individuals, better intellectual control and provide a single point of contact. This metric minimizes the impact of an unbalanced organization, whereby the DMO may be two levels deep but 90% of the total organization reports into that DMO.
Example: In our example this ratio is (7/30)*100. Seven engineers report to ABCA and the org to which ABCA belongs to is of size 30.

6. Level of Organizational Code Ownership (OCO): The percent of edits from the organization that contains the binary

owner or if there is no owner then the organization that made the majority of the edits to that binary.

Implications: The more the development contributions belong to a single organization, the more they share a common culture, focus, and social cohesion. The more diverse the contributors to the code itself, the higher the chances of defective code, e.g., synchronization issues, mismatches, build breaks. If a binary has a defined owner then this measure identifies whether the remaining edits to the binary was performed by people in the same organization (common culture). This measure is particularly important when a binary does not have a defined owner, as it provides a measure of how much control any single organization has over the binary. Also if there is a large PO value due to several of the engineers only having worked on the binary a few times the OCO measure will counter-balance that taking into account the development activities in terms of the edits.

Example: This ratio is 200/ (200+40+10). 200 is the highest proportion of edits made in org reporting to AB. This ratio is computed against the total edits of 200+40+10 across all the three orgs.

7. Overall Organization Ownership (OOW): This is the ratio of the percentage of people at the DMO level making edits to a binary relative to total engineers editing the binary. A high value is good.

Implications: As with previous ownership measures the more the activities belong to a single organization, the more they share a common culture, focus, and social cohesion. Furthermore, the bigger the organizational distance the more chance there is of miscommunication and misunderstanding of goals focus, etc. This measure counter balances OCO and PO to account for a common phenomenon in large teams that exist due to "super" engineers. These engineers have considerable experience in the code base and contribute a substantial amount of code to the system. We do not want one or a few such engineers influencing our measures nor do we want them to be ignored. PO, OCO and OOW account for this type of inter relationship.

Example: In our example we observe that five engineers contributed code reporting to the manager ABCA. There were a total of 32 editing engineers contributing code to this binary across the orgs. Hence the percentage of engineers in org is 5/32.

8. Organization Intersection Factor (OIF): A measure of the number of different organizations that contribute greater than 10% of edits, as measured at the level of the overall org owners.

Implications: Greater is the OIF the more diffused is the contribution to a binary. This implies a lack of strong ownership from one particular org. This measure is particularly important when a binary has no owner as it identifies how diffused the ownership is across the total organization.

Example: In our example, there are totally 250 edits. 10% of this is 25 edits. We observe that all the two organizations under the Master owner (AB, AC) contributed more than 25 edits. Therefore the OIF here is 2. Ideally a lower value is considered to be better.

The measures proposed here attempt to balance the various assertions about how organizational structure can influence the quality of the binary, some of which seem to represent opposing positions. A high level summary of the assertions and the measures that purport to quantify these assertions is presented in Table 1. The measures are motivated more by these concepts and not going bottom-up by fitting all the available data to statistical models.

Table 1: Summary of organizational measures

Assertion	Metric
The more people who touch the code the lower the quality.	NOE
A large loss of team members affects the knowledge retention and thus quality.	NOEE
The more edits to components the higher the instability and lower the quality.	EF
The lower level is the ownership the better is the quality.	DMO
The more cohesive are the contributors (organizationally) the higher is the quality.	PO
The more cohesive is the contributions (edits) the higher is the quality.	OCO
The more the diffused contribution to a binary the lower is the quality.	OOW
The more diffused the different organizations contributing code, the lower is the quality.	OIF

5. CASE STUDY AND RESULTS

In this section we describe our case study and results of our experiments on Windows Vista. Section 5.1 describes our case study set-up and a correlation analysis to identify the inter-relationships between elements discussed in Section 4. Section 5.2 provides an overview of the institutional knowledge in Windows to define and publish a baseline for prior engineer's experience on large legacy projects. Section 5.3 illustrates the building of prediction models using the organizational metrics to predict failure-proneness. Section 5.4 discusses the building of prediction models using other metrics to compare against the model built using organizational measures to predict failure-proneness.

5.1 Description

The organizational metrics defined in Section 4 are collected relative to the release point of Vista. We obtained access to the people management software at Microsoft that maintains employee information like employee ids, email alias, start date at Microsoft. We did not access any personally identifiable information like nationality, age, sex etc. Using this information we built a tree map of the organization structure as illustrated by the example in Figure 1. To maintain an appropriate sense of scale for the study we restrict ourselves to the analysis of Windows Vista. We extracted from the version control system (VCS) for Vista the code check-in information which includes check-in history, date, size of check-in. Our quality variable is defined by post-release failures. Post-release failures are measured for the first six months of the release of the product. All

organizational changes were monitored for Vista development from the beginning milestone for Vista on a fortnightly basis. The overall data was collected for Vista was across 3404 binaries which account for a code size greater than 50 Million Lines of Code (LOC). A discussion of the correlation matrix between the various elements and the inter-relationships between them can be found in an extended version of this paper [28].

5.2 Quantifying Organizational Knowledge

Often large commercial legacy systems have a substantial number of engineers experienced in a prior version of the system who architect and build the new versions. Unfortunately there has been no empirical quantification on the proportion of experienced engineers who work in the new version. The overall motivation of this section is to quantify and publish the proportion of engineers who worked in Windows Vista who had prior experience in an earlier version of Windows. For this purpose we collected organizational, people metrics and code check-in data from the VCS for seven years to quantify the number of experienced engineers working in Windows Vista. Our observations were:

a. 33% of the Vista engineers had contributed code to Windows Server 2003 or Windows XP. This only includes engineers whose code made it into the released version of Windows server 2003 or Windows XP. Engineers that developed tests or process software, for these releases of Windows, are excluded from the analysis.
b. 61% of the engineers had managers who had contributed code to Windows Server 2003 or Windows XP.
c. 37% of them had managers of managers who previously contributed code to Windows.
d. Each legacy binary (i.e. binary that has shipped in an earlier version of Windows) had an average of 31 engineers working on it, of which 2 had check in code on the same binary in Windows Server 2003, 15 had check in code on previous versions of Windows. The remaining 14 engineers had not checked released code into previous versions of Windows but may have worked on other aspects of Windows, or have checked code into other Microsoft products.

These results are across the complete development cycle of Windows Vista in which several thousand engineers contributed code. We plan on using this data to observe if there are any differences in the organizational knowledge/experience for new versions of Windows in the future. We also hope other external companies can use these results to baseline their projects against Windows. An interesting point to note is the significant difference in values between point (a) and (b). This is explained by the fact that over the course of time a number of the engineers who had worked on XP/Server 2003 would have been promoted and would now be managers in Vista, hence the significant difference.

5.3 Predicting Failure-Proneness

In order to determine if organizational metrics defined in section 4 are effective indicators/predictors of code quality we use the eight organizational metrics as predictors in a logistic regression equation to classify Windows Vista binaries as failure-prone or not. Our dataset consists of the above defined 3404 binaries exceeding 50 Million LOC where each binary has its eight organizational metrics and post-release failures mapped. Failure-proneness is the probability that a particular software element (such as a binary) will fail in operation in the field. The higher the failure-proneness, the higher the probability of experiencing a post-release failure. To classify the binaries in Vista in two categories, not failure-prone and failure-prone we define a statistical lower confidence bound (LCB) on all failures. The general form of a logistic regression equation is given as in Equation 1:

$$\text{Probability } (\pi) = \frac{e^{(c+a1*X1+a2*X2+\ldots)}}{1 + e^{(c+a1*X1+a2*X2+\ldots)}} \quad (1)$$

where a1,a2 are the logistic regression predicted constants and the X1,X2…are the independent variables used for building the logistic regression mode. In our case the independent variables are the eight organizational metrics. Binaries with failures lower than the LCB are classified as not failure-prone and other binaries are failure-prone.

An important question to address is whether all the eight organizational metrics are required in building the model. To address this we use two approaches.

(i) Step-wise regression [20]: Step-wise regression is a robust technique compared to normal regression. The initial regression model consists of the predictor having the single largest correlation with the dependent variable. Subsequently, new predictors are selected for addition into the model based on their partial correlation with the predictors already in the model. With each new set of predictors, the model is evaluated and predictors that do not significantly contribute towards statistical significance in terms of the F-ratio are removed so that, in the end, the best set of predictors explaining the maximum possible variance is left. A step-wise regression performed using the eight organizational measures as the predictor variables and post-release failures as the dependent variable did not yield any reduction in the number of predictor variables (**retaining all eight measures**) indicating that all eight organizational metrics were contribute towards explaining the variance in accounting for the post-release failures.

(ii) Principal Component Analysis (PCA) [17]: When the organizational measures are inter-correlated they can suffer from multicollinearity [17] – i.e. over fitting of the data due to inter-correlations between elements which can lead to inflated variance in the prediction of failure-proneness. PCA can account for the multicollinearity among the measures. With PCA, a small number of uncorrelated linear combinations of metrics (that account for as much sample variance as possible) are generated, such that the transformed variables are independent. Running a PCA on the eight organizational measures resulted in the generation of eight principal components indicating that PCA does not reduce the computation overhead in anyway by transforming the organization measures into fewer factors which can be used as predictors.

From the above results of using PCA and step-wise regression we can observe that all the eight organizational measures contribute towards explaining the variance in the post-release failures (our dependent variable) and hence we retain all eight measures to build our logistic regression equation to predict failure-proneness.

We use the technique of data splitting [25] to measure the ability of the organizational measures to predict failure-proneness. The data splitting technique is employed to get an independent assessment of how well the failure-proneness could be estimated from a population sample. We randomly select two thirds (2268) of the binaries to build the prediction model and use the remaining one third (1136) to verify the prediction accuracy. If the predicted failure-proneness probability is > 0.5 we call the binary failure-prone, otherwise not failure-prone.

In order to determine the efficacy of the organizational metrics to predict failure-prone binaries we compute precision and recall of each random split. Zhang and Zhang indicate that due to having a small number of defective modules in a sample, using the *probability of detection* and the *probability of false alarm* measures may lead to impractical prediction models [36]. They suggest using *precision* and *recall* measures to measure the accuracy of a software defect prediction model [36]. Table 2 shows the overall classification table produced for each random split to assess how well the actual and estimated classifications for the 1136 binaries match.

Table 2: Classification table

		Predicted	
		Not Failure-prone	Failure-prone
Actual	Not failure-prone	**a**	**b**
	Failure-prone	**c**	**d**

Precision (*P*) is computed as the proportion of the predicted failure-prone binaries that were correct, as calculated using the equation 2:

$$P = \frac{d}{b+d} \quad (2)$$

Recall(R) is the proportion of failure-prone binaries that were correctly identified, as calculated using the equation 3:

$$R = \frac{d}{c+d} \quad (3)$$

Theoretically a perfect prediction has a precision=recall=1. In order to address the issue of generalizability we repeated the random split 50 times to ensure validity of the experiment. Figure 2 and 3 show the respective precision and recall values for the 50 random splits. The average precision across all 50 random splits was 0.87 or 87% of the predicted failure-prone binaries that were actually failure-prone. Similarly the average recall value was 0.84, i.e. 84% is the proportion of failure-prone binaries that were correctly identified. Also to quantify the sensitivity of the prediction we ran a Spearman's rank correlation between the predicted failure-proneness probability values of the 1136 binaries and actual post-release field failures. The more positive and stronger the correlation the greater the sensitivity indicating that binaries with a high value of predicted failure-proneness have a high number of actual field failures. We repeated the correlation for all the 50 random splits. Figure 4 shows the correlation values. All the correlations were strong, positive and statistically significant at 99% confidence indicating that as the predicted failure-proneness increased the actual field failures also increased.

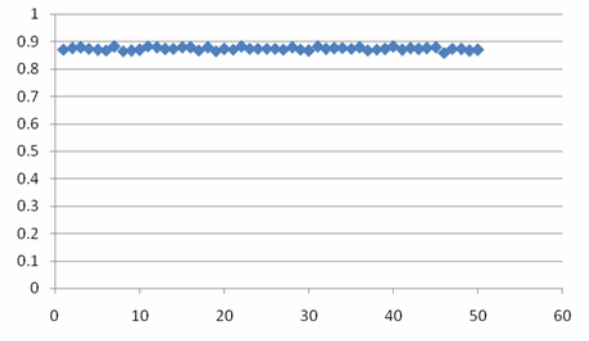

Figure 2: Precision values for 50 Random splits

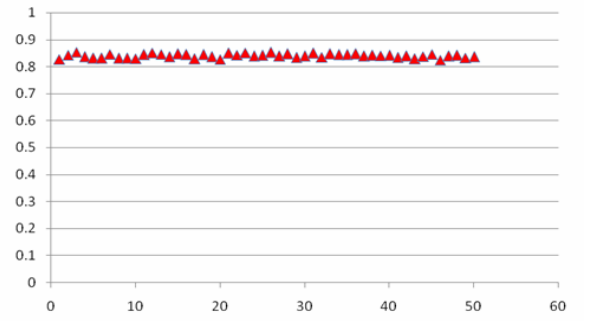

Figure 3: Recall values for 50 Random splits

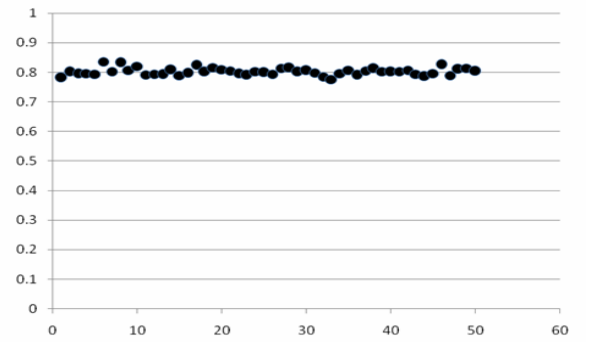

Figure 4: Spearman correlation for 50 Random splits

From the average precision and recall values and the correlation results across 50 random splits along with the consistency of the predictions evident from Figure 2, 3 and 4 across a large sample of 3400+ binaries spread across 50 Million LOC we can observe the efficacy of the organizational metrics to predict failure-proneness in Windows Vista. The precision, recall values and strength of correlations computed are substantially better than our

previous results [24-27] and recent results on predicting failures in large systems [35].

5.4 Comparing Prediction Models

Prior research in software engineering has traditionally used code churn, code complexity, code coverage, code dependencies and pre-release defect measures to build prediction models to identify failure-prone/fault-prone binaries. A summary of previous work in these five areas was presented in Section 2. To compare how the model built using organization metrics performs against models built using code churn, code complexity, code coverage, code dependencies and pre-release defect measures we collect five sets of these measures at the release point of Windows Vista. The data described below is used to build five comparable models. The granularity of measurement of all data is at the level of binaries across the Windows code base and post-release failures are mapped back as earlier to each binary.

(i) Code Churn Model: A total of three churn measures are collected.

a. Total Churn: Total added, modified and deleted lines of code of a binary occurring during the development of Vista. The churn is measured relative to Windows Server 2003, the latest release before Vista.
b. Freq: The number of time that a binary was edited during its development cycle. The implication is the greater the number of edits then the greater the risk of failures.
c. Repeat Freq: The number of consecutive edits that are performed on a binary. A consecutive edit is when the binary is edited during build N and build N+1 and then between builds N+1 and N+2. This is a measure of the instability of the binary during its development, the greater the repeat frequency then the greater the instability of the binary during its development.

(ii) Code Complexity Model: A maximum (Max – at a function/module level) and total (Total – at a binary level) of eight complexity metrics to a total of 19 complexity values are collected per binary. Windows code is predominantly non-OO (Object-Oriented). Only Binaries that have OO code have the OO metrics shown below measured. (Denoted by the notation– (if any)).

a. (Max)(Total) Cyclomatic complexity[21] measures the number of linearly-independent paths through a program module.
b. (Max)(Total) Fan-In: # of functions calling function *f*()
c. (Max)(Total) Fan-Out: # of functions called by function *f*()
d. (Max)(Total) Lines of Code (LOC)
e. (Max)(Total) Weighted methods per class (if any)
f. (Max)(Total) Depth of Inheritance (if any)
g. (Max)(Total) Coupling between objects (if any)
h. (Max)(Total) Number of sub classes (if any)
i. Total Global variables.

(iii) Dependencies Model: For code dependencies we measured both data dependence and call dependence including dependence information at the function level, including caller-callee dependencies, imports, exports, RPC, COM, Registry access, etc. For each binary (e.g. A.dll) we compute the following dependency metrics.

a. Incoming direct : Number of incoming direct dependencies to a binary A.dll
b. Incoming closure: Number of incoming indirect dependencies to a binary A.dll
c. Outgoing direct: Number of outgoing direct dependencies to a binary A.dll
d. Outgoing closure: Number of outgoing indirect dependencies to a binary A.dll
e. Layer information: The architectural layering of Windows computes the distance of a binary from the system hardware (CPU) i.e. the Kernel.

(iv) Code coverage Model: We use the total block and arc coverage measures (analogous to statement and branch coverage) for each binary within Windows Vista.

a. Block coverage
b. Arc coverage

(v) Pre-release defects Model:

a. Number of pre-release bugs found during Vista's development, prior to its release.

We use the computation of precision and recall as defined in Section 5.3 to determine the efficacy of identifying failure-prone binaries across the entire system (3404 binaries, 50 + Million LOC). Table 3 below shows the precision and recall value of the classification models built using the five different types of data sources compared to the organizational metrics model. From Table 3 we observe that organizational structure metrics are significantly better predictors for identifying failure-prone binaries in terms of precision, and recall compared to models built using code churn, code complexity, code coverage, code dependencies and pre-release defect measures. For example comparing between the precision for the models built by organizational metrics and code churn we observe a difference of 7.6% which across our sample size translates to several hundred more binaries being correctly classified.

Table 3: Overall model accuracy using different software measures

Model	Precision	Recall
Organizational Structure	**86.2%**	**84.0%**
Code Churn	**78.6%**	**79.9%**
Code Complexity	**79.3%**	**66.0%**
Dependencies	**74.4%**	**69.9%**
Code Coverage	**83.8%**	**54.4%**
Pre-Release Bugs	**73.8%**	**62.9%**

6. THREATS TO VALDITY

Internal validity. In our study internal validity issues primarily deal with the causal issues of our results. These concerns are addressed to some extent due to the fact that the engineers in Windows had no knowledge that this study was being performed for them to artificially modify their behavior/coding practices or organizational structure to affect our measurements. Further, two of the authors belong to Microsoft Research, an organization outside of Windows and the third author is not a Microsoft employee. Hence there is no internal motivation to show results either way to influence Windows. The experiment does suffer from experimenter bias. To minimize this bias the second author worked independently to collect and process the data and the first and third authors independently analyzed the results.

Construct validity. Construct validity issues arise when there are errors in measurement. This is negated to an extent by the fact that the entire data collection process of the organizational metrics, failures and VCS is automated. These concerns are also alleviated to some extent by the cross check among the organizational measures to identify abnormal values for any of the measures (in addition to the manual checking of the dataset for inconsistencies if any), and by the large size and diversity of our dataset.

External validity. External validity issues may arise from the fact that all the data is from one software system (albeit one with many different binaries) and that the software is very large as other software systems used for a similar analysis may not be of comparable size. To address this issue the analysis was replicated on a reduced set basis to determine the type and size of organization structure required to be able to use the same approach. The results indicate that a team of size 30 engineers and three levels of depth should be sufficient to collect the organizational metrics to predict failure-proneness. We plan to replicate this experiment in smaller organizations at Microsoft to explore this further.

7. CONCLUSION AND FUTURE WORK

In this paper we have reported on our empirical investigation of organizational metrics from the perspective of software quality. We define a set of organizational measures that quantify the complexity of a software development organization. The organizational measures are then used to quantify and study the effect that an organization structure would have on software quality. More generally, it is beneficial to obtain early estimates of software quality (e.g. failure-proneness) to help inform decisions on testing, code inspections, design rework, as well as financial costs associated with a delayed release. Our organizational measures predict failure-proneness in Windows Vista with significant precision, recall and sensitivity. Our study also compares the prediction models built using organizational metrics against traditional code churn, code complexity, code coverage, code dependencies and pre-release defect measures to show that organizational metrics are better predictors of failure-proneness than the traditional metrics used so far.

Drawing general conclusions from empirical studies in software engineering is difficult because any process depends to a large degree on a potentially large number of relevant context variables [3]. For this reason, we cannot assume a priori that the results of a study generalize beyond the specific environment in which it was conducted [3]. Researchers become more confident in a theory when similar findings emerge in different contexts [3]. Since this study was performed on the Windows operating system we plan to replicate this study in other organizations within and outside of Microsoft. There has been preliminary interest in collaborating with the Fraunhofer research institute to replicate this study in other companies. At Microsoft groups have begun investigating the organizational structure of teams to study their impact on quality.

We also plan to investigate this line of research from the context of open source teams where virtual organizations exist to quantify appropriate organizational measures and study their effect on quality. Two other areas of future work is to study such organizational metrics in the context of global software development [13], and collaborate with cognitive psychologists and organizational behavior researchers to look at social and cognitive aspects of our work by doing observational studies of engineers. We also plan to compare our empirical quantification of the organizational experience/knowledge of Windows to compare and contrast with data from other companies to supplement the observational studies. A more detailed empirical discussion of the development of the metrics via GQM [2] is also planned.

Acknowledgements

We would like to thank Windows senior management for giving us access to the employee information database. We would like to thank Tom Ball and Jim Larus for discussions and feedback on our work.

REFERENCES

[1] V. Basili, Briand, L., Melo, W., "A Validation of Object Oriented Design Metrics as Quality Indicators", *IEEE Transactions on Software Engineering*, 22(10), pp. 751 - 761, 1996.

[2] V. Basili, G. Caldiera, and D. H. Rombach, "The Goal Question Metric Paradigm," in *Encyclopedia of Software Engineering*, Vol. 2: John Wiley and Sons, Inc., pp. 528-532, 1994.

[3] V. Basili, Shull, F., Lanubile, F., "Building Knowledge through Families of Experiments", *IEEE Transactions on Software Engineering*, 25(4), pp.456-473, 1999.

[4] S. Biyani, Santhanam, P., "Exploring defect data from development and customer usage on software modules over multiple releases", Proceedings of International Symposium on Software Reliability Engineering, pp. 316-320, 1998.

[5] L. C. Briand, Wuest, J., Ikonomovski, S., Lounis, H., "Investigating quality factors in object-oriented designs: an industrial case study", Proceedings of International Conference on Software Engineering, pp. 345-354, 1999.

[6] F. P. Brooks, *The Mythical Man-Month, Anniversary Edition*: Addison-Wesley Publishing Company, 1995.

[7] S. R. Chidamber, Kemerer, C.F., "A Metrics Suite for Object Oriented Design", *IEEE Transactions on Software Engineering*, 20(6), pp. 476-493, 1994.
[8] M. E. Conway, "How Do Committees Invent?" *Datamation*, 14(4), pp. 28-31, 1968.
[9] T. DeMarco and T. Lister, *Peopleware*. New York: Dorset House Publishers, 1977.
[10] G. Denaro, Pezze., M, "An empirical evaluation of fault-proneness models", Proceedings of International Conference on Software Engineering, pp. 241-251, 2002.
[11] P. Frankl, Weiss, S., "An Experimental Comparison of the Effectiveness of Branch Testing and Data Flow Testing", *IEEE Transactions in Software Engineering*, 19(8), pp. 774 - 787, 1993.
[12] T. L. Graves, Karr, A.F., Marron, J.S., Siy, H., "Predicting Fault Incidence Using Software Change History", *IEEE Transactions in Software Engineering*, 26(7), pp. 653 - 661, 2000.
[13] J. D. Herbsleb, Grinter, R. E., "Splitting the Organization and Integrating the Code: Conway's Law Revisited", Proceedings of International Conference on Software Engineering, pp. 85-95, 1999.
[14] J. D. Herbsleb, Grinter, R. E., "Architectures, coordination, and distance: Conway's Law and beyond", *IEEE Software*, 16(5), pp. 63-70, 1999.
[15] J. D. Herbsleb, Mockus, A., "Formulation and preliminary test of an empirical theory of coordination in software engineering", Proceedings of European Software Engineering Conference/Foundations in Software Engineering, pp. 138-147, 2003.
[16] M. Hutchins, Foster, H., Goradia, T., Ostrand, T., "Experiments of the effectiveness of dataflow- and control flow-based test adequacy criteria", Proceedings of International Conference on Software Engineering, pp. 191-200, 1994.
[17] E. J. Jackson, *A User's Guide to Principal Components*: John Wiley & Sons, Inc., 1991.
[18] T. M. Khoshgoftaar, Allen, E.B., Goel, N., Nandi, A., McMullan, J., "Detection of Software Modules with high Debug Code Churn in a very large Legacy System", Proceedings of International Symposium on Software Reliability Engineering, pp. 364-371, 1996.
[19] T. M. Khoshgoftaar, Szabo, R.M., "Improving Code Churn Predictions During the System Test and Maintenance Phases", Proceedings of IEEE International Conference on Software Maintenance, pp. 58-67, 1994.
[20] D. G. Kleinbaum, Kupper, L.L., Muller, K.E., *Applied Regression Analysis and Other Multivariable Methods*. Boston: PWS-KENT Publishing Company, 1987.
[21] T. J. McCabe, "A Complexity Measure", *IEEE Transactions on Software Engineering*, 2(4), pp. 308-320, 1976.
[22] A. Mockus, Fielding, R.T., Herbsleb, J., "Two case studies of open source software development: Apache and Mozilla", *ACM Transactions on Software Engineering and Methodology*, 11(3), pp. 309 - 346, 2002.
[23] A. Mockus, Zhang, P., Li, P., "Drivers for customer perceived software quality", Proceedings of International Conference on Software Engineering, pp. 225-233, 2005.
[24] N. Nagappan, Ball, T., "Using Software Dependencies and Churn Metrics to Predict Field Failures: An Empirical Case Study", Proceedings of International Symposium on Empirical Software Engineering, pp. 364-373, 2007.
[25] N. Nagappan, Ball, T., "Use of Relative Code Churn Measures to Predict System Defect Density", Proceedings of International Conference on Software Engineering, pp. 284-292, 2005.
[26] N. Nagappan, Ball, T., Murphy, B., "Using Historical In-Process and Product Metrics for Early Estimation of Software Failures", Proceedings of International Symposium on Software Reliability Engineering, pp. 62-74, 2006.
[27] N. Nagappan, Ball, T., Zeller, A., "Mining metrics to predict component failures", Proceedings of International Conference on Software Engineering, pp. 452-461, 2006.
[28] N. Nagappan, Murphy, B., Basili, V., "The Influence of Organizational Structure On Software Quality: An Empirical Case Study," Microsoft Research Technical Report (http://research.microsoft.com), MSR-TR-2008-11, 2008.
[29] L. J. Osterweil, "Software Processes Are Software Too", Proceedings of International Conference on Software Engineering, pp. 2-13, 1987.
[30] T. Ostrand, Weyuker, E., Bell, R.M., "Predicting the location and number of faults in large software systems", *IEEE Transactions in Software Engineering*, 31(4), pp. 340 - 355, 2005.
[31] T. J. Ostrand, Weyuker, E.J, Bell, R.M., "Where the Bugs Are", Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 86-96, 2004.
[32] D. Parnas, "On the Criteria to be Used in Decomposing Systems into Modules", *Communications of the ACM*, 15(2), pp. 1053-1058, 1972.
[33] D. E. Perry, Staudenmayer, N. A., Votta, L., "People, Organizations, and Process Improvement", *IEEE Software*, 11(4), pp. 36-65, 1994.
[34] A. Pogdurski, Clarke, L.A., "A Formal Model of Program Dependences and its Implications for Software Testing, Debugging, and Maintenance", *IEEE Transactions in Software Engineering*, 16(9), pp. 965-979, 1990.
[35] A. Schröter, T. Zimmermann, and A. Zeller, "Predicting Component Failures at Design Time," Proceedings of International Symposium on Empirical Software Engineering, pp. 18-27, 2006.
[36] H. Zhang, Zhang, X., "Comments on "Data Mining Static Code Attributes to Learn Defect Predictors"", *IEEE Transactions in Software Engineering*, 33(9), pp. 635-636, 2007.
[37] T. Zimmermann, Weißgerber, P., Diehl, S., Zeller, A., "Mining Version Histories to Guide Software Changes", *IEEE Transactions in Software Engineering*, 31(6), pp. 429-445, 2005.