Organizational Volatility and its Effects on Software Defects

Audris Mockus
Avaya Labs Research
233 Mt Airy Rd, Basking Ridge, NJ
audris@avaya.com

ABSTRACT

The key premise of an organization is to allow more efficient production, including production of high quality software. To achieve that, an organization defines roles and reporting relationships. Therefore, changes in organization's structure are likely to affect product's quality. We propose and investigate a relationship between developer-centric measures of organizational change and the probability of customer-reported defects in the context of a large software project. We find that the proximity to an organizational change is significantly associated with reductions in software quality. We also replicate results of several prior studies of software quality supporting findings that code, change, and developer characteristics affect fault-proneness. In contrast to prior studies we find that distributed development decreases quality. Furthermore, recent departures from an organization were associated with increased probability of customer-reported defects, thus demonstrating that in the observed context the organizational change reduces product quality.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*Product metrics*; D.2.8 [**Software Engineering**]: Metrics—*Process metrics*; D.2.8 [**Software Engineering**]: Metrics—*Software science*; D.2.9 [**Software Engineering**]: Management—*Programming teams*; D.2.9 [**Software Engineering**]: Management—*Software quality assurance*; K.4.3 [**Organizational Impacts**]: [Employment]

General Terms

Management, Measurement, Reliability

Keywords

Organizational volatility, software defects

1. INTRODUCTION

The key premise of Organization is that it allows more efficient production [20], including production of software. Organizational design defines roles, processes, and formal reporting relationships to improve the functioning of an organization. Our investigation is focused on studying how the organizational change affects software development. Past work investigating software organizations found that effective organizations have very low worker turnover [7] due to what authors call "social capital" invested in the workforce through culture of trust and respect, generous benefits, and recognition of importance of people's personal lives. It is, therefore, of interest to quantify the benefits of low turnover or costs of high volatility.

We have borrowed and extended measures of organizational change based on archival records as considered in, for example, Geisler [9]. Our focus on individual developers lead us to study organization at the lowest level. We operationalize organization for an individual developer as their immediate supervisor based on the formal reporting structure [1]. From the individual's perspective that we take here, we measure three primary organizational events: the arrival of new colleagues into an organization; the departure of colleagues from the organization; and the change of the developer's organization.

The primary positive results of organizational volatility would be the innovation brought by incoming people [19]. The negative side involves the overhead for the existing team to train new developers, the initial lack of experience of new developers, and the gaps in tacit knowledge produced by the departure of experienced developers. While the effectiveness of an organization is sometimes considered through the effectiveness of collaborative work, we are looking at the effects of organizational change at a personal level. After all, collective strategies can only improve upon the individual performance, but if the individuals are not performing, the collective strategies are not likely to help.

The primary effect of an organizational change on product quality is likely to come through the volatility's impact on an individual developer and on the loss of expertise that may lead to otherwise preventable mistakes. Establishing a relationship between the organizational change and product quality faces several challenges. First, product quality has been shown to be affected by a number of factors. Therefore, we adjust for the factors known to affect software quality before investigating the potential impact of organizational change. Second, the organizational change and product quality may be simultaneously influenced by external factors reflecting, for example, the business environment at a particular time. To alleviate this problem we investi-

FSE-18, November 7–11, 2010, Santa Fe, New Mexico, USA.

[1]We also considered alternative operationalizations based on organization's name, and they lead to similar results as noted in the text.

gate the quality of different parts of a system over different time periods.

We propose three individual-centric measures of organizational volatility: proximity to organizational change; arrival of new colleagues; and departure of existing colleagues. We formulate hypotheses describing the effects of volatility on fault proneness and evaluate them using a large software project involving several hundred developers.

The work contributes in scientific, methodological, and practical dimensions.

1. From the scientific perspective we propose and evaluate measures of organizational change, establish that they increase the probability of defects in our context, and reproduce results from several prior empirical studies of software quality.
2. From the methodological perspective we propose a way to evaluate different parts of the system at different moments in time to reduce or eliminate the impact the business environment might have on a single release and propose the need to evaluate the impact of novel factors such as organizational change and distributed development only after adjusting for factors that have been shown to affect software quality.
3. From the practical perspective we quantify the effects of organizational change on the probability of customer-reported defects and establish their relative importance in comparison to predictors derived from code, change, social structure, and geographic distribution. This can guide further investigations and quality improvement efforts towards areas of software and software development that are the biggest contributors to software defects.

We start by discussing the related work in Section 2 and present the context of the study and data collection in Section 3. The hypotheses are presented in Section 4, organizational change measures in Section 5, operationalizations of software quality predictors in Section 6, and results in Section 7. Limitations of the study are outlined in Section 8 and conclusions are presented in Section 9

2. RELATED WORK

A substantial number of research- and practice-oriented publications are devoted to organizational change. While much of it is action oriented and prescriptive, little is devoted to methods to measure organizational volatility from archival data, despite Webb and colleagues [21] raising the issue as early as 1966 well summarized by a Chinese proverb: "the palest ink is clearer than the best memory." A notable exception is the work of Geisler [9] that proposes a number of organizational change measures based on archival records. Unlike the measures we use that are based on human resource systems, Geisler's work required a manual search through archived documents.

A particular form of volatility related to downsizing has received substantial attention. In particular, a study of mid-level managers and executives in Canadian civil service by Armstrong-Stassen [3] found that the downsizing has been associated with a reduction in productivity of the survivors (individuals that remain), an increase of their workload demands, an increase in escape coping strategies, and an increase in the incidence of health problem symptoms. The study used four surveys — one prior to, two during, and one after the downsizing over a three year period. The four survey response rates were 58, 46, 44, and 43 percent correspondingly. The analysis distinguished control-based coping strategies that included positive thinking (recasting the downsizing event as a challenge), direct-action (focusing more energy on your job), and instrumental support (seeking information from others), and escape coping strategies that included the avoidance of the situation and the disengagement (putting less effort into work and spending more time on activities outside work). Other empirical evidence also shows that the downsizing is associated with decreased job performance, for example, Amabile and Conti [1]. In addition to the focus on software developers, unlike in prior work, we also look at more general types volatility than a simple downsizing. Furthermore, we look at a full sample of developers without an inconvenience of high non-response rates and our observations cover the entire period without restrictions to survey administration times.

The most relevant work on software quality is a multiple case study by Cataldo and colleagues [6] that, unlike previous studies, has quantified relative contribution of module size, and syntactic, logical, and social factors. It found that customer reported defects in two distinct systems (not related to the system under study) are predominantly explained by logical coupling (defined by Modification Requests (MRs) touching several files), followed by social coupling (defined via workflow on resolving MRs) of the developers making changes to that file. Little or no impact on customer reported defects was explained by call or data flows. We adjust for predictors used by Cataldo et al. in the model before adding factors indicating organizational volatility. We also report the relative contributions of predictors in explaining the observed variance in quality.

Static organizational factors have been shown to affect software quality in, for example, Nagappan and colleagues [17]. Authors constructed eight measures related to organization for Windows Server binaries, fit a logistic regression model to predict failure proneness, and found that its predictions are similar or better than for other models involving code churn, code complexity, call and data flow, code coverage, and pre-release bugs. Unlike work by Cataldo and colleagues [6] it does not combine information from code and organization into a single model or even report the direction or size of the impact of individual predictors.

Finally, we also consider factors related to distributed development. In particular, Mockus [14] showed that work transferred across locations halved the productivity. However, Bird and colleagues [4] found that the quality of binaries in Windows Vista was not related to offshore development, though it also does not combine information from code and organization into a single model.

3. CONTEXT AND DATA SOURCES

We investigate a large switching software development project in Avaya with several hundred developers. The product is the call processing software installed on many Avaya telephony systems. It embodies several decades of knowledge and experience in the telephony field. In a recent release, the software contains approximately nine million lines of code mostly in C and C++. The software development organization deploys major releases on a fixed schedule, with subsequent minor releases that bundle patches and refinements to the system.

Two primary sources of data were utilized in the study.

The changes to the source code were obtained from the (custom-built) change management system that has been used over more than eighteen years. The data were cleaned to eliminate the administrative changes (changes made for the purpose other than to enhance or fix the product). For example, the initial deltas for branches that do not change the source code were excluded. The data cleaning and validation was done to support project measurement and prior studies, for example, [15] and, therefore, only the essential details are described here. In addition to version control data we also obtained information from problem tracking and customer support systems to find changes related to customer reported issues and to calculate developers' workflow relationships graph based on the tasks they transfer to each other. The workflow graph was constructed by tracking the ownership history of each modification request (MR). The nodes of the graph represent individuals. MR's history of ownership contains timestamp-individual-attribute tuples representing individuals modifying MR's attributes, e.g, making status or priority changes and assignments. The timestamp-developer-attribute tuples are chronologically ordered for each MR and any two individuals immediately subsequent in this order are represented by a link in the workflow graph.

The second source of data was an organizational database (POST) that lists individuals, their organization, and their contact information. We had collected frequent snapshots of this data over a period of seven years. The primary purpose of this data was to establish the identity of each developer and to calculate measures of organizational volatility. As with any other source of data, it had its share of anomalies and issues. First, developer logins were not always identical to email handles in POST. Second, logins have changed over time for some developers because a recent policy required logins to match email handles. Third, the email handles and even organizational IDs have changed for some developers. For example, offshore developers who started at a US location and later went back to their permanent offshore site got a new ID at their home site. To deal with these issues we used a NIS database (snapshots of which we have also collected over seven years) that mapped login to the organizational ID and the full name of the person authorized to use the login. This extra piece of information allowed us to establish the identities of developers over time despite changes in the organizational IDs, email handles, and sometimes even names (for example, as a result of a marriage). We have used these sources of data to map logins and organizational IDs to unique numeric IDs identifying each participant. These unique IDs were then substituted for logins in the code change data and for organization IDs in the POST data to normalize identity information and to provide more privacy (some developers could be recognized from their login). While there were more than ten thousand entries in POST over the considered period, only around two thousand of them were developers who modified the source code in Avaya and 480 of them have modified code in the product we investigate.

It's worth noting the way in which we have prepared these data sources that were obtained in June, 2009. The version control and configuration management data goes back to 1993, and the organizational reporting structure back to 2001. For our analysis we reconstruct the state of the data at each moment in time (at the resolution of calendar weeks). For example, the size of a file at time t is calculated as the size of the file after the last modification during the week that includes time t. If the file was not modified during that week, the size of the file after the first change preceding that week is used. Similarly, logical and workflow dependencies for a file at time t are calculated based only on the historic information that was available at time t. Exact calculations are shown in Table 2.

Furthermore, to capture the concept that the activities performed on a file may lead to introduction of defects at a later time we separate the data into the 12-month *measurement period* and the immediately subsequent 12-month *prediction period*. We use the *prediction period* only to measure the customer reported defects, while the *measurement period* (and, for some predictors as noted in Table 2, all prior history) is used to obtain quality predictors.

The innovative feature of the division into prediction and measurement periods is that these periods are chosen separately for each file to avoid being biased by peculiar circumstances of a single software release. More details are in Sections 8.

4. HYPOTHESES

To hypothesize the effects of organizational change on software quality we borrow from a variety of theoretical frameworks and empirical results. In general we would like to observe if in our context the following holds:

Proposition 1. *Organizational volatility reduces the quality of the software product.*

This outcome would be expected based on a variety of research results discussed in Section 2. More specifically,

Proposition 2. *New experienced members would bring innovations and, therefore, find new ways to improve quality.*

The manner in which the volatility affects quality depends on a variety of factors and theoretical assumptions. In particular, the arrival of new experienced members is likely to increase quality through innovations they may bring [19].

Proposition 3. *New inexperienced members would be more likely to introduce defects.*

As was shown in, for example, [16], changes done by developers with less experience were more likely to introduce a defect. New developers would be less familiar with the system and more likely to make mistakes leading to defects.

Proposition 4. *Outgoing members would leave gaps in the tacit knowledge, making suboptimal design and implementation decisions more likely by the remaining team. This would increase the probability that defects will be introduced or not found prior to release.*

The departure of experienced members may leave gaps in the tacit knowledge, see, for example, Nonaka [18]. Such gaps, may lead the remaining team members to make suboptimal or even disastrous design and implementation decisions. Furthermore, if such departures are related to unfavorable business conditions and downsizing, that may exert additional stress on the remaining employees thus reducing their performance for reasons discussed in Section 2.

Factors that were shown to affect defects.

Even though our main focus is to establish the relationship between organizational change and software defects, we included a variety of predictors known to affect software

Table 1: Concepts and their operationalizations.

Concept	Operationalization
Proximity in time to the organizational change	Time (in years) until the next and after the last change in the organization ID
Size of the reorganization	Number of employees leaving the organization over past two months
New recruits	Number of employees entering the organization over past two months
Size of the organization	Number of employees within the organization
Other factors	Product, Location, Organization ID, Developer ID

quality to ensure that organizational volatility is not confounded with any of these predictors. Each predictor has an associated hypothesis of how it may affect software quality. We present only the essential reasoning because the detailed reasoning for each may be found in prior work and is beyond the scope of the paper. More discussion is in Section 6

Proposition 5. *Larger files will have lower quality.*

Size is the most commonly used predictor in quality studies and is invariably associated with lower quality (all other factors being equal).

Proposition 6. *Files modified by diffused changes and files with high logical coupling will have lower quality.*

Change diffusion (the number of modules or files a change modifies) was found to be associated with an increased probability that a change will contain a fault [16]. Logical coupling of the file (file being changed together with other files), often indicates non-explicit dependencies among files and have been shown to be related to the increase in probability of defects in, for example, Cataldo et al [6].

Proposition 7. *Files modified by developers who have complex workflow will have lower quality.*

High numbers of edges in the workflow network of developers working on the file was associated with increased number of defects in, for example, Cataldo et al [6]. It is not unreasonable to assume that the size of developer's social network (found to increase the defect prediction accuracy by Bird et al [5]) is also positively associated with defects in Windows binaries, even though the direction of the effects or the explanatory power of the predictors were not presented. Finally, an association between large in-degree of a workflow network and lower productivity was reported in, for example, [11].

Proposition 8. *Files modified by developers with low project experience will have lower quality.*

Low project experience of developers working on the module was associated with the higher probability of defects in, for example, [16]. This is also related to Proposition 4.

The following propositions consider factors that have not been found to reduce software quality but were related to other adverse outcomes.

Proposition 9. *Files modified by developers from multiple development sites will have lower quality.*

Distributed development was associated with longer MR resolution times in, for example, Herbsleb and colleagues [12]. Bird and colleagues [4] compared binaries modified from multiple sites in Microsoft and found a nearly significant relationship between the offshore location and higher number of defects. Often developers in offshore sites have less experience with the legacy code or system than developers in sites that have originally created it. The issues of coordination may also increase the potential for defects to be introduced in distributed development contexts.

Proposition 10. *Files modified by changes that are incorporated into multiple releases will have lower quality.*

Such modifications indicate that the issue causing the change is serious enough to affect multiple releases. It also indicates additional dependencies associated with changing code in multiple releases simultaneously. Therefore, files that contain more such modifications may be more fault-prone. The relationship between changes affecting multiple releases and defects has not been investigated but Herbsleb and colleagues [11] found that such MRs have longer cycle times.

5. MEASURES OF ORGANIZATION AND ITS CHANGE

We followed Geisler [9] in measuring organizational change. However, our data and our hypotheses were substantially different, therefore the operationalizations of the measures also have little resemblance. One of the factors that we considered was the reporting hierarchy of the organization to capture changes in the organizational structure brought by a reorganization. The staff reductions discussed above are also often associated with the reorganization.

The measures we propose and evaluate are calculated for developer-time pair (l, t) and are listed in Table 1. For example, Armstrong-Stassen [3] found that the problems associated with the change were transient and did not manifest themselves either well before or well after the downsizing. Therefore, our first measure is the proximity to the organizational change. Denote $O(l, t)$ to be a function that specifies the organization for Developer l at Time t. We operationalize proximity to the organizational change in two ways:

1. Time (in years) until the subsequent change
$$P_{next}(l,t) = \arg\min_{s>0} O(l, t+s) : O(l, t+s) \neq O(l, t)$$
2. Time (in years) from prior change
$$P_{prior}(l,t) = \arg\min_{s>0} O(l, t-s) : O(l, t-s) \neq O(l, t)$$

We also anticipate that the extent of the reorganization measured by the number of people coming in and leaving the organization should have effects described in Propositions 2, 3, and 4. The measure counting the newcomers is needed to investigate Proposition 3, and the number of employees leaving the organization is needed for Proposition 4. In particular, for Developer l at Time t, the number of colleagues who left the organization is calculated as $L(l,t) = \aleph\{p : O(p, t-\delta) = O(l,t) \wedge O(p,t) \neq O(l,t)\}$, where p denotes a person, $\aleph$ denotes cardinality, and δ represents two months. The number of newcomers is obtained in a similar manner $N(l,t) = \aleph\{p : O(p, t-\delta) \neq O(l,t) \wedge O(p,t) = O(l,t)\}$.

The organizational unit was operationalized in two ways: through a supervisor ID and through an organizational ID. The results were similar for both of these operationalizations. In Table 1 we present only one of the operationalizations. Alternative operationalization may be obtained by replacing words "supervisor ID" by "organizational ID."

In addition to measures needed to test our proposition we need to adjust for the systematic variations among developer organizations. Therefore, we look at the number of employees in the organizational unit developer belongs to. The size of the organization for a developer l at time t is simply $S(l,t) = \aleph\{p : O(p,t) = O(l,t)\}$.

6. QUALITY PREDICTORS

When testing a hypothesis about a phenomenon it is important to account for factors that are known to have an impact. For example, if we study how organizational change affects software quality, the model needs to contain the software complexity and other properties affecting software quality. Otherwise, if the organizational change happens to be high only for the most complex parts of software, we may end up ascribing effects of software complexity to organizational change. At the same time, we can not include all possible predictors in a model, because many predictors are strongly correlated among themselves. From the practical perspective, we want to quantify the relative impact of an entire class of quality predictors to understand the nature of the quality problems and the extent to which they can be addressed by improvements in that dimension. Therefore we group quality predictors into several classes and pick a subset of predictors in each class that are not strongly correlated among themselves.

The choice of predictor classes was based on effects reported in the literature and, in particular, based on the several prior studies we sought to reproduce. In particular, they mirror the hypotheses in Section 4. We have grouped quality predictors into code, change, developer, organization, and geographic distribution predictors. The predictors shown in Table 2 and described below were calculated at the individual file level based on the changes made in the past 12 months (*the measurement period*). The response was an indicator that at least one defect for that file was reported by a customer in the subsequent 12 months (*the prediction period*). Because the measures (for example, the file size) often vary over the 12-month period, we used the largest value observed over that period. While code-derived measures are calculated directly at the file level, other measures were aggregated over developers or changes as noted. Instead of averaging, we used the maximum observed value over the changes (and developers making changes) during the 12-month *measurement period*. The primary reason for choosing extreme instead of average values was that "a rotten apple spoils the barrel." In other words, a single risky change or an inexperienced developer may increase the chances of introducing a defect.

Organization-derived measures.

Nagappan and colleagues [17] have used several novel predictors based on organizational structure. The basic premise of these predictors was the organizational ownership of the part of a system for which defects are predicted. Owner organization was defined as the organization lead by the lowest-level supervisor who's subordinates made at least 75% of the modifications to a file. The level of that supervisor (the length of the reporting chain from the top executive to that supervisor of the owner-organization) was referred to as "Depth of Master Ownership (DMO)" in [17] and it was argued that the greater depth allowed the organization to focus on the code, speed-up the decision making, and allow intellectual control leading to fewer defects. It also proposed a number of other predictors based on the concept of organizational ownership. For our data none of them were statistically significant predicting files with defects after adjusting for code, change, and social factors, so we omit a more detailed description of these measures. It is considered in more detail in Section 7.3.

More generally, the concept of organizational ownership of a file appears to be somewhat limited. After all, individual developers are making changes to a file, and organization affects code only through these developers, not directly. Thus, we tried a variety of developer-centric organizational measures. To account for the size of the organization, we used the size of the organization defined by the immediate supervisor of a developer who made changes to the file.

The organizational change measures we chose were based on Propositions 2 and 4 and are described in Table 1. Because there were multiple developers (each often making multiple changes), we calculated the maximum of each measure over all developers making changes during the *measurement period*. We made exception for the measures of proximity to the organizational change (where we use the smallest value) because of the evidence suggesting the transience of the effects of such changes as suggested by, for example, Armstrong-Stassen [3].

Code-derived measures.

Code size and complexity, as well as syntactic dependencies arising from call- and data-flow graphs are typical examples of source code measures. Larger and more complex files were more likely to have a defect in numerous prior studies. There are a number of object-oriented predictors that were shown to correlate with defects (see, e.g., [2]), but the product under consideration uses C language. We have chosen the number of non-commentary source code lines (LOC) in a file to use as the code related predictor. The other code-derived measures, such as call- and data-flow and various code complexity measures were highly correlated with file size, therefore we have not included them in the model together with LOC.

Change-derived measures.

Logical dependencies are represented by changes that modify several files simultaneously and have been found to explain the most variance in the observed defects (see, for example, [6]). The basic assumption underlying this measure is that a logical change requiring modifications to more than one file implies that the decisions involving that file depend in some way on the decisions involving other files. It is similar to logical coupling discussed by Gall and colleagues [8]. We used the simplest logical dependency measure: the number of other files that have been modified together with the file being measured. The increase in logical dependencies has been shown to increase the fault proneness, therefore we expect the same outcome in our study as stated in Proposition 6.

In the product under consideration, changes could be submitted to multiple releases. The product supported multiple releases in the field in addition to at least two releases under

Table 2: Quality predictors. Tuple (l, f, mr, t, r) is an abbreviation for "File f was modified by Developer l at Time t as a part of MR mr and that MR was submitted to Release r." Tuple (l_1, l_2, t) denotes an edge in the workflow graph constructed prior to t. The *measurement period* is $[t-1, t]$.

Class	Predictor	Description
Organization	Size of organization	The maximum size of the organization over developers modifying the file during the *measurement period*: $\max_{l:(l,f,t-1\leq t_o\leq t)} S(l, t_o)$
	Time from prior change	The minimum time from the prior organizational change over developers modifying the file during the *measurement period*: $\min_{l:(l,f,t-1\leq t_o\leq t)} P_{prior}(l, t_o)$
	Time until next change	The minimum time until the next organizational change over developers modifying the file during the *measurement period*: $\min_{l:(l,f,t-1\leq t_o\leq t)} P_{next}(l, t_o)$
	Number leaving org.	The maximum number of people leaving organizations over developers modifying the file during the *measurement period*: $\max_{l:(l,f,t-1\leq t_o\leq t)} L(l, t)$
	Number of newcomers	The maximum number of people coming into organizations over developers modifying the file during the *measurement period*: $\max_{l:(l,f,t-1\leq t_o\leq t)} N(l, t)$
File	LOC	Lines on non-commentary source code (maximum over the *measurement period*)
Change	Logical Deps.	The number of other files changed by the past MRs modifying the file: $LD(f,t) = \aleph\{f_o : \exists mr, \exists t_1, t_2 \leq t,\ (f_o, mr, t_1) \wedge (f, mr, t_2)\}$
	Release Deps.	The maximum number of releases an MR is submitted to over MRs modifying the file during the *measurement period*: $R(f,t) = \max_{mr} \aleph\{r : \exists t_o \leq t,\ (r, mr, t_o)\}$
	Change Diffusion	The maximum number of files changed by an MR modifying the file during the *measurement period*: $D(f,t) = \max_{mr} \aleph\{f_o : \exists t_o \leq t,\ (f_o, mr, t_o)\}$
Social	Workflow	The maximum degree of the workflow network over developers modifying the file during the *measurement period*: $W(f,t) = \max_l \aleph\{l_o : \exists t_1 \leq t, \exists t_2 \in [t-1, t],\ (l, f, t_2) \wedge (l_o, l, t_1)\}$
	Years of prj. experience	The minimum of the years of experience over all developers modifying the file during the *measurement period*.
Geography	Distributed development	The number of sites that modified the file during the *measurement period*
	Mentor Offshore	The maximum of the indicator that a mentor is in another site over developers modifying the file during the *measurement period*

development. Defect fixes had to be submitted to all still-maintained releases the defect affects. It is reasonable to assume that defects affecting multiple releases may be more important than defects affecting only a single release. This release-dependency appears to be different from logical dependencies described in the prior paragraph and, therefore, may serve as another measure of fault proneness. The fix implemented on a file during the *measurement period* that affected the largest number of releases was used to obtain the number of releases.

The change diffusion (the number of files modified by a change) was shown to be a good indicator of the probability that a software patch will fail in [16]. Unlike logical dependencies that are calculated over the entire history of the file, we look only at changes made during the *measurement period* and pick the one with the largest diffusion. The change diffusion is used to generate logical dependency measure, but it captures more transient events. A particularly complex change involving numerous files that was done over the *measurement period* may increase the probability of a defect during the subsequent *prediction period*.

Developer-derived measures.

Workflow (social-network) measures provide information about coordination needs (see, e.g., [11]). Workflow predictors reflect the task assignment of developers working on the code (see, for example, [6]) and are typically calculated through a history of individuals creating, assigning, resolving, and confirming a defect or all individuals who participate in modifying the defect status, including comments. We use the simplest predictor: the number of other individuals that a developer working on a file has interacted with while resolving defects. The wider network of interactions was associated with increased fault proneness in [6], therefore we expect a similar outcome in our study.

Increased developer experience reduced the probability that a patch will fail in [16]. We measured developer experience by calculating the total time in years since the developer made the first code modification in the project. We expect that the increase in developer experience would reduce the fault proneness. We used the developer with the lowest experience who made changes to a file during the *measurement period*.

Geography-distribution-derived measures.

As with developer-derived measures for each source code file we also derive geographic distribution measures based on developers who modified that file. We count the number of distinct geographic locations for developers modifying a file during the *measurement period*. We expect it to increase the fault-proneness because of potential coordination problems among developers in multiple locations.

Because the 12-month *measurement period* may not fully reflect the history of the file we also obtain mentors for each developer making changes during the *measurement period*. The mentors for each developer (follower) are obtained using the method described in [14]. The basic idea is to find developers who have modified the most similar set of files the follower has modified, but earlier. As reported in [14], the cases of mentorship crossing geographic boundaries have roughly

Table 3: Summaries of the model predictors. ± indicates (un)supported proposition or (un)reproduced study. The effects column shows increase in the modeled probability of defects from the doubling (adding 1 if 0) of the median value for that predictor while keeping the remaining predictors at their median values.

Class	Predictor	Effect	Propstns	Reproduced	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
Org. chng	Size of org.	38%	control	+[17]	1	11	22	21.53	37	225
	From prior (yrs)	-15%	+1	new	0	0.38	1.11	1.26	2.42	7.58
	Until next (yrs)	-4%	+1	new	0	0.31	0.77	0.71	1.55	5.49
	Left	26%	+4	new	0	0	1	1.16	3	149
	Newcomers	N/A	−3,2	new	0	0	0	0.48	1	146
File	LOC	34%	+5	+various	1	86	145	157.06	261	43914
Change	Logical Deps.	11%	+6	+[6, 5]	0	0	1	3.42	18	2627
	Release Deps.	192%	+10	+[11]	1	1	1	1.65	2	9
	Change Diffusion	6%	+6	+[16]	1	144	614	458.90	1928	4419
Social	Workflow Deps.	35%	+7	+[6, 5, 11]	0	34	148	73.96	262	293
	Experience (yrs)	18%	+8	−[16]	0	0.73	3.60	3.35	9.37	15.78
Geo.	Distributed	15%	+9	−[4],+[12]	0	0	0	0.17	0	1
	Mentor offshore	69%	+9	new	0	0	0	0.03	0	1

halved the productivity of the follower as compared to instances where mentorship did not cross geographic boundaries. It is reasonable to expect that the drop in productivity may have something to do with the less efficient transfer of expertise to an offshore location. Therefore it is reasonable to expect that this reduced expertise would also have a negative impact on quality.

7. RESULTS

First we present the model quantifying the relative impact of organizational change and other factors on the probability of customer reported software defects. Then we provide basic summaries of the predictors used in the model and, finally, we discuss issues we encountered while trying to reproduce prior studies on software quality.

7.1 Customer-reported defect model

Table 4 presents the estimated logistic regression coefficients, their standard errors, p-values, and deviances (the part of the observed variance explained by the coefficient) for the model predicting defects reported by customers over the 12 month *prediction period* based on changes made to a file over the preceding 12 month *measurement period*. Table 3 summarizes model predictors, the increase in probability from doubling the median value of the predictor, the evidence for our 10 propositions, and the reproducibility of prior work.

Organizational change measures.

Shorter times until the next or since the last organizational change are associated with higher fault proneness (FP) (in Table 4) supporting Proposition 1. Doubling of the median values decreases the probability of defects by 15% and 4% correspondingly (in Table 3).

The only predictor that is not statistically significant is the number of newcomers to the organizations, thus Propositions 2 and 3 do not get support.

It appears that the number of individuals leaving the organization may create gaps in knowledge reducing quality as suggested in Proposition 4. This predictor also explains a substantial amount of variance in FP and its doubling (to two) increases FP by 26%.

The increase in the size of the organization the developer belongs to is associated with lower quality, presumably because of the difficulties of coordination, slower decision making, and reduced intellectual control as hypothesized in, e.g., [17]. However, organizational predictors as a group explain only two percent of the variance after adjusting for other factors. The doubling of organization's size from the median value (to 44) increases FP by 38%.

Because Propositions 2 and 3 provide conflicting predictions about the impact of newcomers this result is not particularly surprising. Furthermore, the one year *prediction period* may be insufficient for newcomers to be allowed to make significant enough modifications that may adversely affect customers.

Change and file measures.

The large files are more fault-prone, supporting Proposition 5. Change-related measures have the largest impact on fault proneness explaining 26% of the variance (more than the remaining predictors combined) supporting Proposition 6, and 10 and reproducing results in [6, 16, 11]. Notably, doubling the number of releases an MR is submitted to from the median value of one almost triples the FP. The files that are touched with many other files, that contain changes submitted to multiple releases, and that contain changes spanning many files are more likely to be fault-prone. The only surprise is that the release dependencies appear to be even more important than logical dependencies.

Workflow measures.

Increase in developers' workflow complexity increased the fault proneness supporting Proposition 7 and reproducing results in [5, 6]. Doubling of the median workflow increases the modeled FP by 35%. The workflow impact is somewhat smaller (it explains less than two percent of the variance) than observed in [6], perhaps reflecting the much larger size of the project than projects considered in [6]. The code complexity and logical coupling may take a more significant role for older and larger projects. Even though we expected the developer experience to be associated with lower fault-proneness as stated in Proposition 8 (the proposition was not supported and the result in [16] was not reproduced), the opposite result is not particularly surprising. In many projects more experienced developers are assigned to more critical and difficult tasks. For example, Zhou and colleagues [22] quantified how fast developers move toward more critical

tasks over time. Thus, the high experience of developers who modify a file may simply reflect the difficulty, complexity, and fault-proneness of that file.

Geographic distribution measures.

Both, the number of geographic locations and the remoteness of developer's mentor increased fault proneness, suggesting that, at least in this study, the geographic distribution has a negative impact on software quality. Thus we get support for Proposition 9 and support for MR cycle-time study [12]. However, we could not reproduce Bird and colleagues' [4] work reporting no impact of distributed development on software quality. The geographic distribution, while significant, explains less than one percent of the variance after adjusting for other factors (7% of the variance before the adjustment). The work from multiple sites increases fault-proneness by 15% and having a remote mentor by 68% as shown in Table 3.

7.2 Summaries of predictors

Table 3 presents summaries of the predictors used in in Table 4. All except the two indicator variables are transformed via natural logarithms (by first adding 1) in the model. Therefore, for the transformed variables Table 3 presents geometric means: $e^{\text{Mean}(\log(X+1))} - 1$. The median file size is 145 non-commentary source code lines and median logical dependencies are with just one other file. While one MR is included in nine different releases (and affects 36 files), more than half of the files have been modified by MRs included in at most one release. The median of the maximum diffusion is quite high: 614. The median of the maximum workflow dependencies shows that half of the files were modified by a developer who has encountered at least 148 other developers. 17% of the files have been modified from multiple locations and only 3% had a mentor at another site. The reporting structure shows a median of 22 people in the maximum of the organization size over developers modifying the file. The median of the minimum time from prior organizational change was 1.1 years and to the subsequent change was 0.77 years. The geometric average of the maximum number of colleagues who left over the prior two months was 1.16 and for newcomers was 0.48.

Table 4: Logistic regression coefficients for the file fault-proneness. Deviances explained by each predictor are in the right column. There are 32099 files, 7% of them contain defects, and 41% of the total deviance is explained by the model.

Class	Predictor	Est.	StdErr	p-val	Devnc
File	log(LOC)	0.43	0.03	0.00	2450
Chng	log(Logical)	0.25	0.02	0.00	978
	log(Releases)	2.67	0.07	0.00	2331
	log(Diffusion)	0.08	0.03	0.00	321
Socl	log(Workflow)	0.43	0.05	0.00	255
	log(Experience)	0.28	0.04	0.00	13
Geo	Distributed	0.14	0.07	0.04	41.94
	Mentor	0.53	0.12	0.00	27.97
Org	log(OrgSize)	0.48	0.06	0.00	160
	log(From)	−0.40	0.07	0.00	51
	log(Until)	−0.06	0.03	0.09	6
	log(Left + 1)	0.33	0.04	0.00	74
	log(New + 1)	−0.01	0.04	0.70	0

7.3 Reproducibility

We have reproduced the main results reported by Cataldo and colleagues [6] on relative contribution of module size, logical coupling, and workflow. We also found that the customer reported defects were predominantly explained by change-derived measures, though in our case the release dependencies were even more important than logical dependencies. Another slight difference was that the workflow factors had a slightly lower impact than in the study by Cataldo.

The measures based on the organizational ownership introduced by Nagappan and colleagues [17] were not statistically significant after code, change, workflow, and organizational change derived predictors were added to our model. However, if we include just the predictors reported in [17], we obtain a reasonable model with a reasonable predictive performance for our project. In addition to the organizational structure, Nagappan and colleagues [17] used the number of past modifications and the number of developers making modifications in the same prediction model. As discussed in Section 8, just the number of past modifications provides an excellent prediction results on our data (approximately 79% sensitivity and specificity) and in other work (see, e.g., [10, 13]). Therefore it is possible that the reasonable prediction performance in [17] also relied on these two predictors. Because the work in [17] provides no information about the direction or the size of the effects, it was not possible to investigate reproducibility in more depth. It is worth noting that measures defined in [17] are quite involved, assume static organizational structure, and appear to be somewhat Microsoft-specific making it difficult to apply them in other contexts. For example, Depth of Master Ownership measure (described above) was defined based on the reporting distance from the supervisor responsible for the entire development, not the company's CEO. In Avaya, some of the changes are made by developers in the services organization, making the CEO the only link between them and the developers in the product organization. Also, because our organization was quite dynamic, we had to calculate organizational ownership measures for each week and aggregate them for the *measurement periods*.

However, we feel that organizational measures do deserve a closer look and we used some ideas proposed in [17] to construct predictors that were applicable in a more dynamic organizational structure studied by us. Instead of looking at the level of the supervisor of the owning organization, we looked at the longest reporting chain from that supervisor to the leaf developer making changes to the file. The longer that distance, the harder it would be for developers to coordinate their decisions because they may need to go through more layers of the reporting chain. The number of individuals in the owner organization was highly correlated (> 0.9) to its reporting depth, so we did not consider it.

The study on prediction of patch failures [16], reported that lower developer experience expressed as the number of past changes to project's code increased the probability that a patch will fail. We observed the opposite result (even though we use the duration of time a developer spent on the project, not the number of past changes, both factors were strongly correlated), perhaps reflecting the observation that senior developers are assigned more critical tasks as shown in [22]. However, the primary driver of patch failure reported in [16] was the diffusion of code changes comprising

the patch. We also found that the change diffusion increased fault proneness.

Bird and colleagues [4] found that the quality of binaries in Windows Vista was not related to offshore development. In contrast, we do find that the files changed by developers with remote mentors or by developers from multiple sites have increased fault-proneness. It is entirely possible that our project was different from Windows Vista considered by authors. However, the basis of the claim in [4] is a fault-proneness model which contains the number of developers making changes to a binary and the indicator of the distributed development having a p-value of 0.056 (not significant at 0.05 level). If we fit an identical model on our data (we predicted for a single release to enable a direct comparison to [4], otherwise the p-value was virtually zero, corresponding to t-statistic of more than 9) we do find that the indicator of distributed development has a p-value of 0.048 (significant at 0.05 level). There is no reason to assume that such a small difference in p-values indicates contradicting results just because of the convention that 0.05 is a magical number determining what is significant. A meta-analysis of both studies may conclude that there is a support for the hypothesis that distributed development does increase fault proneness. Furthermore, the number of developers reflects multiple dimensions of software development and that may make it difficult to identify the individual contribution of each dimension as described in the next section.

8. LIMITATIONS

While we considered the change of the organization's ID for a developer and the arrival and departure of other employees from that organization, there are many other types of organizational change that may affect software quality. In particular, we do not distinguish between a developer moving to another project versus the organization changing its ID.

The two-month period we considered for the purpose of counting the arrival and departure of employees, may not be sufficient to assess the impact of departures as the consequences may manifest over longer periods of time.

The absolute value of Spearman correlations among predictors did not exceed 0.57 (for the correlation between the project experience and the proximity to the prior organizational change), thus collinearity issues are not likely.

We tried to avoid release-specific effects by including information about files at a variety of calendar times. Therefore, for each file we ordered modification times in the period from 2004 and 2008 and chose 75-th percentile of these modification dates (to get a point in time when the file is still being changed — "alive"). A release date closest to that time was then picked for the file and the *measurement period* was based on the 12 months preceding the date and the response was obtained based on the 12 months following it (*prediction period*). We chose 12 month periods because most of the work for a release is completed within a year prior to the release date and most of the defects reported by customers are fixed during a year following the release date.

In a real prediction model we would choose the same moment in time (present) for all the files to separate the *measurement period* and the *prediction period*. We, therefore, considered how the model described in Table 4 would fare in such a setting. To accomplish that we fit it for a fifty randomly selected subsets of files each containing three fourths of the entire population of files and predicting the faulty files in the remaining one fourth of the files. Both, the average sensitivity and specificity were reasonably high (approximately 75%). It is important to note that the sensitivity and specificity are much lower when predicting rare events (in our case, the probability of a defect was less than 0.07). As we hypothesized, each release is unique in many ways. For comparison, for the *measurement period* used to fit the model in Table 4, both the sensitivity and specificity averaged above 84%.

It is important to note that the simplest change-derived predictor is the number of modifications made to a file. As was argued by Cataldo and colleagues [6], the number of past modifications incorporates many factors that affect quality because files that are more complex, less well designed, have logical or other dependencies, or are more fault prone for other reasons, are more likely to be modified. Therefore, even though past changes can predict fault proneness [10, 17], including past changes in the model may make it impossible to separate individual contributions made by code, developer, or organizational factors. The number of developers that made modifications to a file is another obvious predictor. The higher number of developers, the higher is the need to coordinate among them, and that may lead to introduction of defects. However, that number was strongly correlated with the number of modifications to a file (0.94 Spearman correlation). Therefore, we did not include the number of developers making modifications to a file or the number of past modifications in the fault proneness models, despite their excellent predictive power. If the number of past modifications is included as a predictor in the model, the contributions of logical dependencies and change diffusion become non-significant and the estimates of the remaining coefficients and their p-values slightly change (but keep their sign).

9. CONCLUSIONS

The organization's volatility as measured by the proximity to an organizational change was observed to increase the probability of customer reported defects in the context of a large software system.

The influx of new members into the organization had no impact on software quality, possibly because the new developers are not assigned to important changes [23].

On the other hand, departures from organization were associated with the higher probability of customer reported defects, possibly because of the gaps in knowledge and experience left by the departing members.

The larger size of organization was associated with a higher probability of defects possibly reflecting the increased need for coordination and reduced speed of decision making in larger organizations.

We were able to reproduce the results from several prior studies suggesting maturity of the empirical software engineering. We found the results to be quite similar in an exact replication (when we tried to reproduce all aspects of the analysis) even for quite dissimilar projects. However, some differences emerged once we assumed our perspective of quantifying the relative size of various factors affecting the fault proneness. For example, we found the organizational ownership measures introduced in [17] to be difficult to obtain because the organization under study was quite dynamic and because some of the concepts did not have

an obvious correspondence between the organizations in two studies. We also found that a study investigating distributed development and fault proneness [4] reached opposite results because of a minor difference in the observed p-value.

From the research and practical perspective the main result is that organizational change is associated with lower software quality, though the size of the effects follows after change- and code-derived factors in their ability to explain fault-proneness. The dominant factors, confirming prior studies, are related to dependencies embedded in the code through changes. It is worth noting that the project's ability to reduce organizational volatility may be much greater than its ability to reduce the complexity of the logical dependencies, thus addressing organizational issues may be the easiest approach to improve quality.

It is not clear to what extent the organizational volatility causes or is a cause of the file, change, and workflow properties that are associated with high fault-proneness. In particular, the organizational change measures explain more than 20% of the variance in the fault-proneness before adjusting for other factors. In other words, does the organizational volatility lead to low quality code or does the low quality code induce volatility in the organization maintaining it?

More detailed investigations of the nature of organizational change and the nature of quality reductions observed in this study are likely to provide more detailed practical recommendations on ways to organize and reorganize software development. We expect these finding to add an important piece to a puzzle involving the understanding of how the dynamic relationship between the product and the organization affects key software engineering outcomes.

10. REFERENCES

[1] T. Amabile and R. Conti. Changes in the work environment for creativity during downsizing. *Academy of Manegement Journal*, 42:630–640, 1999.

[2] E. Arisholm and L. C. Briand. Predicting fault-prone components in a java legacy system. In *ISESE*, pages 8 – 17, 2006.

[3] M. Armstrong-Stassen. Coping with downsizing: A comparison of executive-level and middle managers. *International Journal of Stress Management*, 12(2):117–141, 2005.

[4] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy. Does distributed development affect software quality? an empirical case study of windows vista? In *ICSE 2009*, 2009.

[5] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy. Putting it all together: Using socio-technical networks to predict failures. In *ISSRE 09*, Bengaluru-Mysuru, India, 2009.

[6] M. Cataldo, A. Mockus, J. A. Roberts, and J. D. Herbsleb. Software dependencies, the structure of work dependencies and their impact on failures. *IEEE Transactions on Software Engineering*, 2009.

[7] D. J. Cohen and L. Prusak. *In Good Company: How Social Capital Makes Organizations Work*. Harward Business Press, 2001.

[8] H. Gall, K. Hajek, and M. Jazayeri. Detection of logical coupling based on product release history. In *ICSM*, pages 190–198, 1998.

[9] E. Geisler. Organizational change phenomena, managerial cognition, and archival measures: Reconceptualization and new empirical evidence. TR 99-02, School of Business, Illinois Institute of Technology, 1999.

[10] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy. Predicting fault incidence using software change history. *IEEE TSE*, 26(2), 2000.

[11] J. Herbsleb and A. Mockus. Formulation and preliminary test of an empirical theory of coordination in software engineering. In *FSE'03*, Helsinki, Finland, October 2003. ACM Press.

[12] J. D. Herbsleb and A. Mockus. An empirical study of speed and communication in globally-distributed software development. *IEEE TSE*, 29(6):481–494, June 2003.

[13] T. M. Khoshgoftaar and N. Seliya. Comparative assessment of software quality classification techniques. *Empirical Software Engineering*, 9(3):229–257, September 2004.

[14] A. Mockus. Succession: Measuring transfer of code and developer productivity. In *2009 International Conference on Software Engineering*, Vancouver, CA, May 12–22 2009. ACM Press.

[15] A. Mockus and D. Weiss. Interval quality: Relating customer-perceived quality to process quality. In *2008 International Conference on Software Engineering*, pages 733–740, Leipzig, Germany, May 10–18 2008. ACM Press.

[16] A. Mockus and D. M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180, April–June 2000.

[17] N. Nagappan, B. Murphy, and V. R. Basili. The influence of organizational structure on software quality. In *ICSE 2008*, pages 521–530, 2008.

[18] I. Nonaka. A dynamic theory of organizational knwledge creation. *Organizational Science*, 5(1):14–37, February 1994.

[19] J. Van Maanen and E. Schein. Towards a theory of organizational socialization. In B. Staw, editor, *Research in organizational behavior*, volume 1, pages 209–264. JAI Press, Greenwich, CT, 1979.

[20] F. W. T. F. W. *The Principles of Scientific Management*. Harper & Brothers, 1911.

[21] E. Webb, D. Campbel, R. Schwartz, and L. Sechrest. *Unobtrusive Measures: Nonreactive Research in the Social Sciences*. Rand McNally Colledge Publishing Company, Chicago, IL, 1966.

[22] M. Zhou and A. Mockus. Developer fluency: Achieving true mastery in software projects. In *ACM SIGSOFT / FSE*, Santa Fe, New Mexico, 2010.

[23] M. Zhou, A. Mockus, and D. Weiss. Learning in offshored and legacy software projects: How product structure shapes organization. In *ICSE Workshop on Socio-Technical Congruence*, Vancouver, Canada, May 19 2009.