Organizational volatility and post-release defects: A replication case study using data from Google Chrome

Samuel M. Donadelli, Yue Cai Zhu, Peter C. Rigby
Department of Computer Science and Software Engineering
Concordia University
Montreal, Quebec, Canada
Email: first.last@concordia.ca

***Abstract*—The quality of software projects is affected by developer turnover. Mockus studied organizational volatility in the context a large switching software project at Avaya. We replicate his model of the impact of organizational volatility on post-release defects. At the time of Mockus's study, Avaya was experimenting with outsourcing and layoffs were prevalent. In contrast, we study volatility on the Chrome web-browser, which is growing rapidly in terms of popularity and team size. Where possible, we use the same factors as Mockus: the number of co-owners, the number of developers joining and leaving the organization, the number of co-changing directories, developer experience and, instead of LOCs, the churn. Our investigation is conducted at the directory instead of the file level. The control variables, including churn, number of co-owners, and expertise all conform with the consensus in the literature that more changes, more co-owners, and lower expertise lead to an increase in customer reported post-release defects. After normalizing by the highly correlated number of co-owners, the number of developers who leave and join both reduce the number of post-release defects. We discuss this unexpected result.**

I. Introduction

As developers join and leave a project, the organization of that project is effected. For example, new developers might bring innovations, while departing developers will leave knowledge gaps that other team members will have to fill. Our goal is to understand how changes to the development team affect software quality. Mockus [8] conducted a study on a single project at Avaya and determined that developers leaving the project had a small negative impact on software projects. In contrast, newcomers did not have a statistically significant impact on quality.

Our goal is to replicate Mockus's study in the context of Google Chrome. During Mockus's study period, Avaya was going through turnover in the form of layoffs and outsourcing [7]. In contrast, Chrome has seen drastic growth in terms of popularity and number of developers. This contrasting replication allows us to determine whether Mockus's findings generalize and to understand which variables are important in predicting organizational volatility in the growth context of Chrome.

We use the same response variable as Mockus – the number of post release customer found defects (CFD). However, we make two changes: first, we group measures at the directory level instead of the file level because few files have multiple defects, second, we use a count of CFDs instead of a binary response.

Many studies have modelled bugs, so we included control variables to take into account known strong predictors such as developer expertise as well as our leaver and joiner measures. Table I lists the predictors used in our model and in Mockus's model as well as a description of how each is calculated. Below, we provide a brief conjecture on the influence of each predictor.

Churn: The more changes are made to a software artifact the more defects that are found in it. Larger size generally correlates with complexity and increased bug density [6], [11].

Co-owners: There is evidence that when more developers touch a software artifact there will be more bugs [1], [8].

Leavers: When a developer leaves a project the team losses that developer's knowledge of the system. This can lead to tacit knowledge gaps that can result in defects [8].

Newcomers or Joiners: Newcomers may not be experienced on the project, but they can bring fresh ideas [4]. Since the core team of Chrome are Google employees, they are stringently vetted increasing the likelihood that they will be good additions. Newcomers may, however, introduce defects through a lack of knowledge about the system and will take time from core developers when they ask questions.

Co-changes and change diffusion: The greater the number of co-changing software artifacts the lower the quality of a software is expected to be. The idea of logical coupling was introduced by Gall *et al.*'s [5] who implied that logical coupling relates to the files that change together in a commit. Change

978-0-7695-5594-2/15 $31.00 © 2015 IEEE
DOI 10.1109/MSR.2015.47

TABLE I
QUALITY PREDICTORS FROM MOCKUS'S PAPER. WE ARE UNABLE TO INCLUDE CERTAIN PREDICTORS BASED ON THE AVAILABLE CHROME DATA. ALL PREDICTORS ARE MEASURED PER RELEASE AT THE DIRECTORY LEVEL.

Predictors	Included	Description
Co-owners	Yes	The number of developers who have worked in a directory. Mockus calls this "size of organization".
Time from prior and next change	No	Mockus had information about changes to the development team. We do not have this information for Chrome.
Leavers	Yes	The number of developers who left the project per directory.
Joiners and Newcomers	Yes	The number of developers who joined the project per directory.
Churn	Yes	The sum of the lines of code added and deleted from all files in a directory.
Co-changing directories	Yes	The number of directories that have co-changed with a directory. Mockus calls this "logical dependencies." We assume that the greater the number of directories in a commit the greater the complexity of the change.
Change Diffusion	Yes	The maximum number of co-changing directories in a commit. We drop change diffusion from our model as it is highly correlated with the number of co-changing directories.
Release Dependencies	No	At Avaya, a change could become part of multiple releases, for Chrome, each change is included in only one release.
Workflow	No	We do not calculate the developer workflow network.
Developer Experience	Yes	The minimum number of years of experience across all developers working in a directory.
Distributed Development	No	We are unable to count the number of different offices Chrome developers work in.
Mentor Offshore	No	We do not calculate mentors and cannot know when a developer is working offshore.

diffusion is the maximum number of co-changing directories in a commit. Mockus *et al.*'s [9] found that change diffusion was an important predictor to estimate defects.

Developer experience: The quality of a system is tightly coupled to the experience and expertise of the developers. Further, experienced developers have a detailed understanding of the design and evolution of a system and are less likely to introduce defects [10].

The paper is structured as follows. In Section II, we describe the methodology and data. In Section III, we present and discuss our results. Finally, we present conclude the paper.

II. METHODOLOGY AND DATA

A. Data

Google Chrome was started in 2008. We mined Chrome data from July 2008 to May 2013. It is developed by more than 1000 contributors and it has more than 150K files. A core team of 176 developers have made 80% of the changes to Chrome. Since our goal is to create an explanatory model of the number of post-release defects, we describe the Chrome release process and how we partition each change and bug into a release.

Chrome uses the issue tracker provide by Google Code.[1] Issues include a summary, a detailed description of the issue, and attachments. After an issue is opened, there is a section where we can find specific details such as status, owner, type, priority, release, operational system and release block. We are able to differentiate between bugs and feature requests, and only include bugs in this work. We are able to link bugs to commits, because the developer who fixed the bug records the commit number. The identification of CFDs is important to our explanatory model because all commits have an issue number.

Classify bugs as CFDs: First, we use the information we mined from the bug database. Second, we need to classify which of these bugs are opened by users. The issue form has a field called "Reported by". In this field, we can find the reporter's e-mail and determine whether the he or she is a project member. If the reporter is not a project member, then the bug was opened by a user.

Releases identification: The official release dates for Chrome is reported in Chrome developers web page.[2] Starting at release 5 Chrome transitioned to a rapid release cycle and produces a release every 6 weeks. We use Rahman and Rigby's tool to extract and differentiate development changes from post-release changes [12].

Developers identification: The developers are identified by the email addresses they used to submit the commit. There are some duplicated email addresses. To resolve this issue, we use Canfora *et al.* [2] name aliasing tool. However, the tool had some flaws. For instance, we created a new automatic script to resolve names that had less than 5 characters.

Newcomers and people leaving the organization: For a given developer, we consider the date of his first change to be his starting date, and the date of his last change to be his ending date.

B. Methodology

1) Quality predictors: Mockus *et al.* [8] used release dates as the starting point for looking one year to the past at file

[1]Chrome issues website http://code.google.com/p/chromium/issues/list

[2]Chrome release information available at http://www.chromium.org/developers/calendar

level to calculate the factors mentioned on Table I, instead of LOCs we use directory churn, which is a strongly correlated measure [6]. As a response, Mockus looked one year to the future, trying to identify if a file had at least one CFD for the specific file during the measurement period.

Our model will be different regarding the granularity level. Instead of examining files, we will use directory level. Most files have zero defects, so we group at directory level, and even then the system is very large. The Chrome release process is also more frequent, so we will consider the cycle of development channel (six weeks) for each release to calculate the predictors in Table I.

2) Model selection: Mockus modelled bugs using a logistic regression, either a file had a bug or it did not. We tried the same model, however, the results were not significant because most files do not have bugs leading to a zero-inflated dataset. As result, we grouped bugs at the directory level and used bug counts as the response, instead of binary response. We considered poisson, zero-inflated poisson and quasi-poisson models to fit our data. We dropped the poisson model because there is overdispersion in our data. The zero inflated poisson model was compared with quasi-poisson model using the Vuong test available in R statistical software. The Vuong test selects the better model based on the likelihood ratio and non-nested hypothesis [14]. After testing both models, the quasi-poisson model was superior.

III. Results and Discussion

There is an extensive literature on bug prediction models, so before we create a model with all of our predictors, we evaluate how well churn and the number of co-owners predict post-release defects. These two initial models give us a baseline against which to compare our future models. Our first model only includes the level of activity, *i.e.* the churn, that a directory has undergone. In Table II we can see that this simple model does quite well explaining 45.23% of the deviance. Other researchers have shown that churn is a good predictor of the number of defects in a module [11], [15]. We replicate these findings showing that the more changes that are made to a directory, the more post-release defects it will contain.

In our second model, we used the number of developers who work in a directory, *i.e.* the number of co-owners. We find that this model does even better with 66% of the deviance explained. Our findings support previous research that has shown that the greater the number developers working on a software artifact, the greater the number of defects [1], [8].

The number of co-owners of a directory is highly correlated with all other predictors. This correlation means that the more developers who touch a file, the more churn, the greater the number of leavers, the greater the number of joiners, and so on. For this reason, we decided to normalize the other predictors by co-owners. The resulting model, Model 3 in Table II, explains 71% of the deviance. All predictors are statistically significant with the exception of the number of co-changing directories, which is dropped from further analysis.

TABLE II
Prediction Models for CFDs

	Model 1	Model 2	Model 3
Churn	0.67		0.16
Co-owners		1.28	1.05
Leavers			-3.04
Newcomers			-0.62
Developer Experience			-1.60
Co-changes			*
Deviance Explained	45.23	66	71

($p < 0.001$, except * $p < 1$))

The interpretation of each variable is complicated because a quasi-poisson model has a log-link function and the explanatory model is on log scale. For these reasons, we investigate the rates of change shown in Table III. For instance, a 50% increase in number of co-owners leads to an increase of 54% in the number of post release defects.

Directory churn: Normalizing churn by number of co-owners means that we are modelling the average churn of developers per directory. By tripling the average churn a developer makes, the number of post-release defects increase by 19%.

Co-owners: Increasing the number of co-owners in a directory has a high impact in the CFDs post-release. For instance, a doubling in the number of co-owners in a directory doubles the number of post-release CFDs.

This result adds to the growing consensus that too many developers touching a software artifact can introduce defects in the form of unexpected dependencies [3].

Number of developers leaving the organization: The correlation between the number of leavers and CFDs is positive. However, this relationship is largely influenced by the number of co-owners – the larger the total number of developers, the larger the number of developers who can leave. As a result, we normalized leavers by the number of co-owners, leavers/co-owners. With this normalization, we see that the more people who leave the fewer number of post-release defects reported by customers. This finding is unintuitive and the opposite direct to what Mockus found. We have conjectured that leavers would lead to knowledge loss and to less experienced team members taking over code they are unfamiliar with.

We suggest two possible explanations that deserve future work. First, when a developer leaves the project, the features they were working on may be put on hold, which would lead to fewer changes and fewer defects. Second, existing co-owners

TABLE III
EFFECT OF ORGANIZATION, DIRECTORY, CHANGE AND SOCIAL FACTORS ON THE NUMBER OF CFDS

	Variables	Estimate	10%	50%	100%	200%
Directory Churn	log(churn/co-owners)	0.16	1.53	6.67 [b]	11.66	19.10
Organization	log(co-owners)	1.06	10.63	53.69	108.48	220.42
	log(leavers/co-owners)	-3.04	-25.18	-70.89	-87.88	-96.47
	log(joiners/co-owners)	-0.62	-5.73	-22.20	-34.90	-49.35
Experience	log(experience/co-owners)	-1.60	-14.18	-47.81	-67.10	-82.83

[a] The quasi-Poisson dispersion parameter is taken to be 6.4 (over-dispersion). To make interpretation easier the proportional change at 10% to 200% is shown for each variable.
[b] For example, a doubling in the average churn made by a developer increases the number of post-release defects by 6.67%.

may take over the leaving developers work leading to a more focused set of changes and fewer defects.

Number of newcomers: Increasing the ratio of newcomers/co-owners in a directory decreases the number of CFDs post-releases. By doubling the ratio of newcomers, we see a decrease of 34.9% CFDs post-release. In Mockus's work, the impact of newcomers was not statistically significant. We find preliminary support that newcomers bring positive innovation without increasing the number of defects. One contextual factor related to Chrome is that Google's hiring process is very stringent allowing them to hire highly skilled developers. The impact of newcomers who are less stringently vetted may lead to different results.

Experience: The longer a developer has been on the project, the fewer the number of CFDs that are found in his or her code. For example, doubling the developer experience decreases the CFDs post-release by 27.31%. The more experienced people are in the project, the better the quality of the code is [10].

Co-changing directories: The number of directories that where co-changed did not have a statistically significant impact on the number of post-release bugs. This runs counter to the findings of Mockus and other who found that logical dependencies were important in determining post-release defects [8], [13]. Logical dependencies are usually calculated at the file level and the more coarse grained directory used in this study may have influenced the result.

IV. THREATS TO VALIDITY

This study is a replication of Mockus's [8] study of a switching system at Avaya. The context of our study is drastically different from the outsourcing that existed at the time of Mockus's study. As a result, some of our findings differ from those of Mockus and future replications are necessary to provide a more generalized understanding of organizational volatility and turnover. We were able to replicate most of the measures used by Mockus and feel that other should be easily able to replicate our work. Future work may also want to investigate different levels of granularity, such as directories vs modules.

Concerning the internal validity of our measures, the greatest threat is the reliability of bug data. The Chrome data set is an excellent source because we are able to differentiate internal issues from bugs reported by end users. Internal issues are highly related to churn, while end user bugs tend to be more representative of defects. One difficulty for further replication will be finding projects that allow research to differentiate internal and external bugs.

V. CONCLUSION

In this paper, we presented a contrasting replication of Mockus's research on organizational volatility [8]. We mined the data from Google Chrome project which is growing in team size and popularity and contrasts with the project at Avaya that was experiencing outsourcing.

We used three predictors that have been extensively studied. We found that the number of co-owners in a directory increases the number of CFDs found post-release. This finding adds to the growing consensus that as more developers work on a software artifact there will be more uncoordinated and buggy changes. In terms of churn, *i.e.* development activity, we found that higher churn leads to more complexity and greater post-release defects. We also found that the greater the developer experience, the fewer the number of CFDs post-releases.

After normalizing for the number of co-owners, we found that the greater the ratio of leavers in a directory, the fewer the number of post-release defects. This result is counter-intuitive and needs future work to determine if other factors are at play. We also found that adding new developers to a directory actually reduced the number of post-release defects. While we find this result surprising and deserving of future work, we suspect that Google does a good job of vetting replacement developers.

Our findings add another data point in our understanding of organizational volatility. There is a clear need for other studies in new contexts to strengthen our understanding turnover on software projects.

REFERENCES

[1] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu. Don't touch my code!: Examining the effects of ownership on software quality. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, pages 4–14, New York, NY, USA, 2011. ACM.

[2] G. Canfora, M. Di Penta, R. Oliveto, and S. Panichella. Who is going to mentor newcomers in open source projects? In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 44:1–44:11. ACM, 2012.

[3] M. Cataldo, A. Mockus, J. Roberts, and J. Herbsleb. Software dependencies, work dependencies, and their impact on failures. *Software Engineering, IEEE Transactions on*, 35(6):864–878, Nov 2009.

[4] M. A. Cini. Group newcomers: From disruption to innovation. *Group Facilitation*, 3(2001):3–13, 2001.

[5] H. Gall, K. Hajek, and M. Jazayeri. Detection of logical coupling based on product release history. In *Software Maintenance, 1998. Proceedings., International Conference on*, pages 190–198, Nov 1998.

[6] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy. Predicting fault incidence using software change history. *IEEE Transactions on Software Engineering*, 26(7):653–661, 2000.

[7] A. Mockus. Succession: Measuring transfer of code and developer productivity. In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, pages 67–77, 2009.

[8] A. Mockus. Organizational volatility and its effects on software defects. In *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE '10, pages 117–126, New York, NY, USA, 2010. ACM.

[9] A. Mockus and D. M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2), April 2000.

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

[11] N. Nagappan and T. Ball. Use of relative code churn measures to predict system defect density. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, pages 284–292, New York, NY, USA, 2005. ACM.

[12] M. Rahman and P. Rigby. Release stabilization on linux and chrome. *Software, IEEE*, pages 2–9, March-April 2015.

[13] E. Shihab, A. Mockus, Y. Kamei, B. Adams, and A. E. Hassan. High-impact defects: A study of breakage and surprise defects. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, pages 300–310, New York, NY, USA, 2011. ACM.

[14] Q. H. Vuong. Likelihood ratio tests for model selection and non-nested hypotheses. *Econometrica: Journal of the Econometric Society*, pages 307–333, 1989.

[15] T. Zimmermann, R. Premraj, and A. Zeller. Predicting defects for eclipse. In *Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on*, pages 9–9, May 2007.