2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)

Does UML Modeling Associate with Lower Defect Proneness?: A Preliminary Empirical Investigation

Adithya Raghuraman
Carnegie Mellon Univ.
adithya@cmu.edu

Truong Ho-Quang,
Michel R. V. Chaudron
Chalmers | Gothenburg University
{truongh, chaudron}@chalmers.se

Alexander Serebrenik
Eindhoven Univ. of Tech.
a.serebrenik@tue.nl

Bogdan Vasilescu
Carnegie Mellon Univ.
vasilescu@cmu.edu

***Abstract*—The benefits of modeling the design to improve the quality and maintainability of software systems have long been advocated and recognized. Yet, the empirical evidence on this remains scarce. In this paper, we fill this gap by reporting on an empirical study of the relationship between UML modeling and software defect proneness in a large sample of open-source GitHub projects. Using statistical modeling, and controlling for confounding variables, we show that projects containing traces of UML models in their repositories experience, on average, a statistically minorly different number of software defects (as mined from their issue trackers) than projects without traces of UML models.**

***Index Terms*—software design, UML, software quality, open-source-software**

I. Introduction

Software design is widely accepted as a fundamental step to developing high-quality software [1].

By making designs developers go through a process of reflection, including discussing trade-offs and alternatives, which should result in more thoughtful designs and more maintainable systems [2]. The communication benefits to explicit software design are also well understood: architectural decisions that developers make become well-documented, reducing information loss and potential misinterpretation during system implementation, and facilitating communication among team members and the onboarding of new developers [2]. Both commercial [2] and open-source software developers [3] alike recognize these potential benefits.

Among modeling languages, the Unified Modeling Language (UML) is often viewed as de-facto standard for describing the design of software system using diagrams [3]. In practice, UML is often used in a loose/informal manner (not adhering stricly to the standard [4]). Also UML is used selectively, focusing on important, critical or novel parts.

Still, despite many expected benefits of UML modeling on software development outcomes, the empirical evidence on the matter is scarce. Notable exceptions include a study by Arisholm *et al.* [5], showing through two controlled experiments involving students that, for complex tasks and after a learning curve, the availability of UML models may increase the functional correctness and the design quality of subsequent code changes. There is also work by Fernández-Sáez *et al.* [6] that suggests an overall positive outlook of practitioners towards UML modeling in software maintenance. Finally, we note an empirical study by Nugroho and Chaudron [2] of an industrial Java system, showing that classes for which UML-modeled classes, on average, have a lower defect density that those that were not modeled.

In this paper we study the intuitive and widely held belief that *the use of UML modeling, on average, should correlate with higher software quality*. To this end, we statistically analyse empirical data obtained from of 143 open-source GitHub projects. Many hypotheses about the benefits of UML models on specific software maintenance outcomes have been proposed [7]. However, more generally, one can expect that the mere practice of UML modeling as part of software development indicates a high team- and process maturity and deliberateness that, in turn, should lead to higher-quality code.

In search of evidence [8] to substantiate this belief, we start from a publicly available data set of open-source software projects on GitHub that use UML models [9], and: 1) assemble a control group of GitHub projects not known to use UML models; 2) mine data from the GitHub issue trackers of both sets of projects (using and not using UML models), estimating their defect rates ("bug" issue reports) as a proxy for software quality; and 3) use multivariate statistical modeling to estimate the impact of having UML models on defect proneness, while controling for confounding factors. Our results reveal a small statistically significant effect of using UML models on defect proneness, *i.e.*, *projects with UML models tend to have fewer defects*.

II. Methodology

We designed a quasi-experiment to compare the defect proneness between two groups of open-source GitHub projects: a *treatment* group of projects using UML models, part of a public data set [9]; and a *control* group of projects sampled randomly using GHTorrent [10]. We describe our data collection and analysis process next.

A. Data

As part of a previous study [11], Robles *et al.* [9] released a data set of 4,650 non-trivial GitHub projects,[1] defined as having at least six months of activity between their first and most recent commits and at least two contributors, that use UML models, as identified by a manually-augmented automated repository mining technique. As our operationalization of defect proneness involves mining the projects'

[1]Available online at http://oss.models-db.com

2574-3864/19/$31.00 ©2019 IEEE
DOI 10.1109/MSR.2019.00024

TABLE I
GITHUB SLUGS FOR THE 50 UML PROJECTS IN OUR DATA SET

abb-iss/SrcML.NET	kite-sdk/kite
aegif/NemakiWare	LibrePCB/LibrePCB
asciidocfx/AsciidocFX	longkerdandy/mithqtt
boost-experimental/di	lviggiano/owner
christophd/citrus	lycis/QtDropbox
claeis/ili2db	mbeddr/mbeddr.core
cliqz-oss/keyvi	MvvmFx/MvvmFx
collate/collate	MyRobotLab/myrobotlab
Comcast/cats	Particular/docs.particular.net
cpvrlab/ImagePlay	pipelka/roboTV
crowdcode-de/KissMDA	plt-tud/r43ples
dandelion/dandelion-datatables	Protocoder/Protocoder
djeedjay/DebugViewPP	rbei-etas/busmaster
droidstealth/droid-stealth	robotology/codyco-modules
eProsima/Fast-RTPS	SINTEF-9012/ThingML
Freeyourgadget/Gadgetbridge	smartdevicelink/sdl_core
GeertBellekens/Enterprise-Architect-Toolpack	SpineEventEngine/core-java
GluuFederation/oxAuth	SpoonLabs/astor
gocd/gocd	telefonicaid/fiware-cygnus
HPI-Information-Systems/Metanome	timolson/cointrader
imixs/imixs-workflow	UESTC-ACM/CDOJ
infinidb/infinidb	uwescience/myria
IQSS/dataverse	vicrucann/dura-europos-insitu
kamilfb/mqtt-spy	xamarin/monodroid-samples
kermitt2/grobid	xen2/SharpLang

GITHUB issue trackers (details below), we only include in our *treatment* (UML) group those projects that had at least 30 issues on GITHUB as of March 2018; we determined the threshold empirically after manual exploration of the data, to filter out largely inactive projects. In addition, we identified using Google's `langdetect` library[2] those projects not using English as their natural language, as our operationalization of defect proneness makes expects issue discussions in English. We further filtered out projects that are not primarily written in either C++, C#, or Java (as labeled by GITHUB), the languages traditionally associated with UML. Next, we filtered our projects with fewer than 10 stars on GITHUB and projects in which the gap between the first commit and the first GITHUB issue is more than a year, in an effort to further exclude student homework assignments and largely trivial or inactive projects [12]. Finally, we excluded projects started before 2009 – shortly after GITHUB itself started – such that all remaining projects could have plausibly used the GITHUB issue tracker from the beginning. After all these filters, the treatment group (Table I) consists of 50 UML projects.

Note the relatively small size of the treatment group after applying the different activity-based filters when compared to the size of the original data set by Robles *et al.* [9], and especially when compared to the size of GITHUB. Still, to our knowledge, this is the largest data set on which our research question has ever been studied empirically.

To assemble a *control* group of projects not known to use UML models, we randomly sampled, using the March 2018 version of GHTORRENT [10], projects that satisfied the same criteria (see above), for a total of 93 non-UML projects after filtering; the version we queried is newer than the one used by Robles *et al.* [9], hence we did not consider projects that did not already exist in the older version. Moreover, we further ensured that the randomly selected control projects were not already present in the treatment group.

[2]https://pypi.org/project/langdetect/

TABLE II
BREAKDOWN OF OUR DATA SET BY LANGUAGE

	Java	C#	C++
UML projects	31	6	13
Control group projects	63	10	20

Table II presents a breakdown of our two groups of projects by programming language.

B. Operationalization

Dependent variable. As a measure of a project's defect-proneness and a proxy for its software quality, we estimate the *number of bug-related issues* reported in its GITHUB issue tracker. To identify *bug-related* issues, as opposed to feature requests, tasks, or other types of issues commonly found in open-source issue trackers [13], we developed a *Naive Bayes* classifier [14]; a similar approach was considered by Zhou *et al.* [15]. Our classifier takes as input the title and the body of an issue, and produces one of two labels, *bug* or *not bug*. [3]

To this end, we started with one author coding randomly sampled issues as *bug* or *not bug*, until labelling 100 of each; unclear cases were discussed between two authors, and subsequently resolved. After manual data labelling, we created 20 random splits of our labelled data containing two equally sized train and test sets, trained the classifier on the train set, and computed the accuracy on the test set; each split preserved the balanced nature of the data, *i.e.*, the train and test sets contained 50 *bug* and 50 *not bug* issues each. The average accuracy of our classifier over the 20 random splits is 89%.

Ultimately, we chose the best performing of the 20 classifier instances and ran it on the unlabelled data, *i.e.*, all the issue reports of all the projects in our UML and control groups. Overall, we labelled 29,983 issues as *bug* and 48,579 issues as *non bug* across the 143 (50 + 93) projects in our data.

Independent variables. Our main predictor variable is a dummy *has UML* that distinguishes between the treatment and control groups. In addition, we cloned all the GITHUB repositories locally and computed several variables for co-variates and confounding factors: *project age*, *i.e.*, the number of days between the first and the most recent commit; *primary programming language*, as reported by GITHUB; *number of contributors*, *i.e.*, distinct commit authors in the project's history, after resolving aliases using a script published[4] by Vasilescu *et al.* [16]; *number of commits*, as a proxy for project complexity; *number of stars*, as a proxy for popularity and size of user base; and *programming language*, coded relative to Java as baseline. Moreover, we used Munaiah *et al.*'s [17]

[3]Classifier code and data analysis script available online at https://github.com/adi1234567890/UML-defect-proneness

[4]https://github.com/bvasiles/ght_unmasking_aliases

TABLE III
LINEAR REGRESSION MODEL SUMMARY

	Response: log(num_bug_issues) Coeffs (Errors)	Sum. sq.
(Intercept)	0.40 (0.63)	
log(age + 1)	0.05 (0.09)	19.11***
log(num_commits)	0.53 (0.06)***	58.35***
log(num_contributors)	0.10 (0.09)	9.03***
log(stars + 1)	0.18 (0.05)***	6.62***
test_suite_ratio	−0.24 (0.47)	0.02
comment_ratio	0.26 (0.54)	0.01
has_CI	−0.17 (0.13)	1.38
has_license	−0.61 (0.28)*	2.73*
languageC#	−0.19 (0.20)	2.60
languageC++	−0.33 (0.15)*	
has_UML	−0.30 (0.14)*	2.24*
R^2	0.61	
Adj. R^2	0.58	
Num. obs.	143	

$^{***}p < 0.001$, $^{**}p < 0.01$, $^{*}p < 0.05$

`reaper`[5] to compute: *has CI*, a dummy indicating whether or not the project uses continuous integration; *has license*, defined analogously; *test suite ratio*, the fraction of test lines of code to source lines of code; and *code comment ratio*, the fraction of comment lines of code to all lines of code. These variables control for projects' level of maturity of their practices.

C. Analysis

To test our hypothesis, we build a multiple linear regression model, a robust technology which enables us to estimate the effects of having UML models on defect proneness *while holding the other independent variables fixed*. We diagnose the model for multicollinearity, checking that the variance inflation factor (VIF) remains below 3 [18]; no variables violated the threshold. We further check if modeling assumptions hold, and observe no significant deviation from a normal distribution in QQ-plots and randomly distributed residuals across the range. The model fits the data well, explaining 58 % of the variance (Adj. R^2). In addition to the regression coefficients, we also report the amount of variance explained by each term (the *Sum. sq.* column in Table III), as obtained from an ANOVA analysis; the relative fraction of the total variance explained by the model that can be attributed to a particular variable can be considered as a measure of its effect size.

III. RESULTS AND DISCUSSION

Interpreting the regression summary in Table III, we observe that among the control variables, only the *number of commits*, the *number of stars*, and the *has license* dummy have statistically significant effects at 0.05 level or below. All three behave as expected: larger projects and projects with larger communities (of users and, hence, potential issue reporters) tend to have more bugs reported, other variables held fixed; projects that declare a license tend to have fewer bugs reported.

Regarding programming languages, we note that C# projects are statistically indistinguishable from Java projects, while C++ projects tend to have fewer bugs reported than Java projects, other variables held fixed.

Finally, we note a small (approximately 2 % of the variance explained by the model) but statistically significant effect of UML models: other variables held fixed, projects with UML models are expected to have about 35% (exp(-0.3); note the log-transformed response) fewer bugs reported than projects without UML models.

IV. THREATS TO VALIDITY

We note several threats to validity [19].

Construct validity. We measured defect-proneness by computing the number of bug-related issues in the issue tracker. However, issues and bugs may not map one-one [20], *e.g.*, several issues could pertain to the same bug, or one issue may encompass several bugs. While imperfect, this operationalization is common in the literature, *e.g.*, [21]. We also note that projects may use issue trackers outside of GITHUB, which we did not track, potentially biasing our defect estimates. We tried to alleviate this threat by only considering projects which used the GITHUB issue tracker substantially, thus arguably reducing the risk that they also use external issue trackers. Another construct validity threat stems from not accounting for different types of UML modeling, *e.g.*, sequence vs class diagrams, and lumping the different UML modeling techniques into one group. We leave analysing this distinction for future work.

Internal validity. Given the multiple linear regression technology we used, with controls for known confounding factors, our results should be relatively robust. Still, we note a threat to internal validity from using a Naive Bayes classifier to label issues. In particular, the algorithm works on the Naive Bayes assumption that given the target label, each of the covariates is independent [22], which may not always hold. Finally we also note that we did not run the original UML mining technique on the control group projects to further confirm absence of UML models in their repositories; since the original UML data set was created by comprehensively mining every project from GHTORRENT, we assume that the risk of mislabeling non-UML control group projects is low. However, it must also be noted that there is the possibility of projects having information about design/architecture in *.pdf, .ppt* and other such files which therefore could've been wrongly classified as projects not using UML modeling by [9].

External validity. We note, again, the relatively small size of our data set, which can largely be explained by the small number of open source projects that meet all selection criteria. Moreover, our sample is not representative (by construction) of open-source or GITHUB as a whole. It is unclear without ample future work and replications, beyond the scope of this paper, whether our results can generalize. Another threat to the external validity comes from the dataset used to train the Naive Bayes classifier. Mature projects are known to write thorough issue reports while less mature projects tend to completely ignore or very sparingly make use of the issue tracker [23]. As a result, the classifier is inherently rigged towards learning features from more mature projects.

[5]https://github.com/RepoReapers/reaper

V. Conclusions

Prior work in this area focused on understanding the impact that UML design had on software projects in a qualitative manner. Through this paper, we try to provide a quantitative analysis of the way in which UML modeling of design relates to the defect proneness of the projects. Our work shows that after controlling for confounding factors, projects that use UML experience, on average, a statistically minorly different number of software defects

Future Work. One of the immediate next steps for this work, as indicated by the threats to validity section, is to distinguish between the type of UML modeling in our treatment group projects. In particular, making the distinction between sequence and class diagrams and studying their corresponding correlation with defect proneness, we believe, will further reduce the dearth of research being done in this area. Another future work involves making a distinction between forward designed and reverse engineered projects [24]. Work done by Fernández-Sáez *et al.* [25] shows a positive outlook of practitioners towards the use of forward design but empirical research regarding the correlation between forward/reverse design and defect proneness, once again, is scarce. A third future work involves reproducing the investigation in a new sample of projects but performing a more precise verification of certain variables. The project members may perhaps be also inquired to check whether UML models were used or not and the way in which they were being used. Lastly, we believe that UML modeling directly influences structural aspect of the software architecture/design to some extent. For instance, we would expect the use of UML structure-diagrams to have an influence on metrics such as cohesion, coupling, etc. of the software projects. Our current work opens the door to studying these questions in the near future.

References

[1] C. Larman, *Applying UML and patterns: an introduction to object oriented analysis and design and interactive development*. Pearson, 2012.

[2] A. Nugroho and M. R. V. Chaudron, "The impact of UML modeling on defect density and defect resolution time in a proprietary system," *Empirical Software Engineering*, vol. 19, no. 4, pp. 926–954, 2014.

[3] T. Ho-Quang, R. Hebig, G. Robles, M. R. V. Chaudron, and M. A. Fernandez, "Practices and perceptions of UML use in open source projects," in *International Conference on Software Engineering (ICSE): Software Engineering in Practice Track*. IEEE, 2017, pp. 203–212.

[4] M. Petre, "UML in practice," in *International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 722–731.

[5] E. Arisholm, L. C. Briand, S. E. Hove, and Y. Labiche, "The impact of UML documentation on software maintenance: An experimental evaluation," *IEEE Transactions on Software Engineering*, vol. 32, no. 6, pp. 365–381, 2006.

[6] A. M. Fernández-Sáez, M. R. V. Chaudron, and M. Genero, "An industrial case study on the use of UML in software maintenance and its perceived benefits and hurdles," *Empirical Software Engineering*, pp. 1–65, 2018.

[7] A. M. Fernández-Sáez, M. Genero, and M. R. V. Chaudron, "Empirical studies concerning the maintenance of UML diagrams and their use in the maintenance of code: A systematic mapping study," *Information and Software Technology*, vol. 55, no. 7, pp. 1119–1142, 2013.

[8] P. Devanbu, T. Zimmermann, and C. Bird, "Belief & evidence in empirical software engineering," in *International Conference on Software Engineering (ICSE)*. IEEE, 2016, pp. 108–119.

[9] G. Robles, T. Ho-Quang, R. Hebig, M. R. V. Chaudron, and M. A. Fernandez, "An extensive dataset of UML models in GitHub," in *International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 519–522.

[10] G. Gousios and D. Spinellis, "GHTorrent: GitHub's data from a firehose," in *International Conference on Mining Software Repositories (MSR)*. IEEE, 2012, pp. 12–21.

[11] R. Hebig, T. H. Quang, M. R. V. Chaudron, G. Robles, and M. A. Fernandez, "The quest for open source projects that use UML: Mining github," in *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS '16. New York, NY, USA: ACM, 2016, pp. 173–183. [Online]. Available: http://doi.acm.org/10.1145/2976767.2976778

[12] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining GitHub," in *International Conference on Mining Software Repositories (MSR)*. ACM, 2014, pp. 92–101.

[13] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: how misclassification impacts bug prediction," in *International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 392–401.

[14] I. Rish, "An empirical study of the naive bayes classifier," T. J. Watson IBM Research Center, Tech. Rep., 2001.

[15] Y. Zhou, Y. Tong, R. Gu, and H. Gall, "Combining text mining and data mining for bug report classification," *Journal of Software: Evolution and Process*, vol. 28, no. 3, pp. 150–176, 2016. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.1770

[16] B. Vasilescu, A. Serebrenik, and V. Filkov, "A data set for social diversity studies of GitHub teams," in *International Conference on Mining Software Repositories (MSR)*. IEEE, 2015, pp. 514–517.

[17] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan, "Curating GitHub for engineered software projects," *Empirical Software Engineering*, vol. 22, no. 6, pp. 3219–3253, 2017.

[18] P. D. Allison, *Multiple regression: A primer*. Pine Forge Press, 1999.

[19] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[20] G. Rodríguez-Pérez, A. Zaidman, A. Serebrenik, G. Robles, and J. M. González-Barahona, "What if a bug has a different origin?: Making sense of bugs without an explicit bug introducing change," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '18. New York, NY, USA: ACM, 2018, pp. 52:1–52:4. [Online]. Available: http://doi.acm.org/10.1145/3239235.3267436

[21] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, "Quality and productivity outcomes relating to continuous integration in GitHub," in *Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. ACM, 2015, pp. 805–816.

[22] H. Zhang, "The optimality of naive bayes," *AA*, vol. 1, no. 2, p. 3, 2004.

[23] J. Cabot, J. L. C. Izquierdo, V. Cosentino, and B. Rolandi, "Exploring the use of labels to categorize issues in open-source software projects," in *International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2015, pp. 550–554.

[24] N. Medvidovic, A. Egyed, and D. S. Rosenblum, "Round-trip software engineering using UML: From architecture to design and back," 1999.

[25] A. M. Fernández-Sáez, M. Genero, M. R. V. Chaudron, D. Caivano, and I. Ramos, "Are forward designed or reverse-engineered UML diagrams more helpful for code maintenance?: A family of experiments," *Information and Software Technology*, vol. 57, pp. 644 – 663, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584914001311