Continuous Defect Prediction: The Idea and a Related Dataset

Lech Madeyski

Wroclaw University of Science and Technology,
Faculty of Computer Science and Management,
Wyb.Wyspianskiego 27, 50-370 Wroclaw, POLAND,
Lech.Madeyski@pwr.edu.pl

Marcin Kawalerowicz

Opole University of Technology, Faculty of Electrical
Engineering, Automatic Control and Informatics,
ul. Sosnkowskiego 31, 45-272 Opole, POLAND
and CODEFUSION Sp. z o.o., Armii Krajowej 16/2,
45-071 Opole, POLAND, marcin@kawalerowicz.net

Abstract—We would like to present the idea of our Continuous Defect Prediction (CDP) research and a related dataset that we created and share. Our dataset is currently a set of more than 11 million data rows, representing files involved in Continuous Integration (CI) builds, that synthesize the results of CI builds with data we mine from software repositories. Our dataset embraces 1265 software projects, 30,022 distinct commit authors and several software process metrics that in earlier research appeared to be useful in software defect prediction. In this particular dataset we use TravisTorrent as the source of CI data. TravisTorrent synthesizes commit level information from the Travis CI server and GitHub open-source projects repositories. We extend this data to a file change level and calculate the software process metrics that may be used, for example, as features to predict risky software changes that could break the build if committed to a repository with CI enabled.

Keywords—*mining software repositories, defect prediction, continuous defect prediction, software repository, open science*

I. INTRODUCTION

Identifying defect-prone modules, packages or files has long intrigued researchers (e.g., [1], [2], [3]). However, limited resources and tight schedules common in software development environments have triggered a change of research focus into identifying defect-prone ("risky") software changes instead of files or packages. Our long-term research goal is to propose Continuous Defect Prediction (CDP) practice supported by a tool set using machine learning (ML)-based prediction models and large dataset (collected from both, open source and commercial projects) to predict defect-prone software changes (at the moment limited to success/fail continuous integration outcomes). We refer to our quality assurance practice as "Continuous Defect Prediction", as developers can receive a continuous feedback from the supporting tool we are working on as to whether the latest code change is risky and could break the build if committed/pushed/checked-in to the repository with Continuous Integration (CI) enabled. We are building on top of previous works on classifying software changes either as being clean or buggy, e.g., of Kim et al. [4], Kamei et al. [5] and Yang et al. [6], as well as the TravisTorrent dataset made available by Beller at al. [7], but aim to deliver immediate and continuous feedback for a developer on how risky the most recent code change is, and to embed this feedback mechanism into the software development practice. Actually, this is a follow up to our previous research on Continuous Test-Driven Development [8]

and Agile Experimentation [9]. Hence, we build upon our experience and tools developed so far. In this paper, we share a large dataset used by our prediction models to identify defect-prone software changes.

Every build on a build server can be triggered by one or a set of commits (multiple commits pushed together to the central repository). First we are collecting information about those commits and then we calculate a set of metrics for every file change that took part in a given CI build. Those metrics include for example a number of modified lines (NML) since the last build, build commit local time of day (BCDTL), a number of distinct committers (NDC) involved in that build, a number of revisions (NR) for a given file, last build status (LBS) of the build where the file was involved. We use these metrics as features (predictors/independent variables) to create prediction models where the build result is the dependent variable.

We are working on CDP in a commercial environment on a real life project where we use Jenkins CI build results from Bitbucket on-premise installation as a source of success/fail build indication. We combine this information with data collected from the Git repository. Due to the non-disclosure agreement we are able to share data collected from a large number of open source projects, but not from the commercial project we are involved in. We are sharing the data collected with help the of the TravisTorrent [7] open database project. TravisTorrent synthesizes data from the Travis CI build server with the data collected from GHTorrent – offline mirror of data provided by the API of popular GitHub version control hosting service. The data on TravisTorrent comes from popular open-source projects such as JRuby or Rails. TravisTorrent stores the data on a commit level meaning a commit is the most fine-grained piece of data it contains. In turn, we are working on a file change level. For us every file changed in a commit taking part in a CI build conveys meaningful information. By using this information, we build classification models and use them to predict the outcome of a build before the data is committed into the repository. Prediction models built on the basis of this dataset are beyond the scope of this data paper.

We have collected more than 11 million rows of data for 1265 GitHub projects where more than 30,000 developers were active. We are making this data available for a broader public in hope it will help other researchers interested in defect-prone software change prediction, behaviour of software developers or other areas of software engineering research and practice.

II. Data collection and storage

We are collecting the CDP relevant data from two sources: continuous integration process and version control system (software repository). The data from CI process are gathered from the CI server. We were interested in two pieces of information the CI server can provide us: build result and mark of the commit or commits involved in that build. This information can be obtained from the CI server over an API or from another source, as an associated database, for example. The database usage can help to deal with temporary CI information. The data from the associated database is usually not cleaned as it is the case with the CI build information. It is customary to keep only the last n builds information (e.g., date and time of the build, its result, build logs) to preserve the storage space on the CI server. We use TravisTorrent database to obtain the data on the build results and commits involved in those builds. TravisTorrent synthesizes the information taken from the Travis CI server using its API and GitHub repository data taken from its offline mirror (GHTorrent). We are specifically querying the `travistorrent_11_1_2017` table and using the following columns:

1) git_trigger_commit (hash of the commit which triggered the build),
2) gh_project_name (project name on GitHub),
3) gh_pushed_at (time of the push that triggered the build),
4) tr_status (build result).

We are accessing Jenkins CI build results stored in a Bitbucket database in our CDP work on a commercial project. Both Jenkins and Bitbucket are installed on premise at a company building banking software. Apart from Jenkins CI, we also made it possible to use the TeamCity CI server API to collect the build results. We support the Git source control system, however the approach does not limit us to this particular system by any means. It is possible to extend the approach to: Subversion, Mercurial or other source control systems.

It is worth mentioning that depending on the chosen CI server, the build results enumeration can vary. We are mapping them to three states: success, failure, and unknown (for example if the build was interrupted, ended with a warning or was marked as unstable by the CI server), as shown in Table I.

TABLE I.   Jenkins, Travis and TeamCity build results mapping

Our database	Jenkins	Travis	TeamCity
1 (success)	SUCCESS	passed	NORMAL
0 (failure)	FAILURE	failed	FAILURE
999 (unknown)	NOT_BUILT		ERROR
	ABORTED	errored	WARNING
	UNSTABLE		UNKNOWN

One build on a CI server can be made for one particular commit or for any number of distinct commits. The information on commits involved in a CI build can be obtained from a CI server API or from a database. We are querying the TravisTorrent database to get the information we need (using the `git_trigger_commit` column described earlier in this section) using the following query:

```
1   SELECT git_trigger_commit, gh_project_name, tr_status, gh_pushed_at
2   FROM travistorrent_11_1_2017
3   WHERE gh_project_name = :projectName
4   AND gh_pushed_at IS NOT NULL
5   AND tr_build_id >=
6   (SELECT tr_build_id FROM travistorrent_11_1_2017
7     WHERE git_trigger_commit=:commit_from)
8   AND tr_build_id <=
9   (SELECT tr_build_id FROM travistorrent_11_1_2017
10    WHERE git_trigger_commit=:commit_to)
11  ORDER BY tr_build_id DESC
```

The specific commits involved in the build are calculated based on the branch topology tree of the software repository.

Then the software repository is utilized as the source of the file level metrics gathered for CDP. Those metrics are the features for the prediction model. In that regard the type of the software repository (whether it is Git, Subversion, Mercurial or any other) from which the data were harvested is irrelevant. We are currently acquiring the data from Git repositories (GitHub to be specific). We are not using GitHub API to avoid problems with bandwidth throttling reported in [10], as in the case of TravisTorrent. We are cloning all of the GitHub repositories to a local disk instead. As a source of data we are using the commit history stored locally. As it turns out, this poses no problem to GitHub and is not a problem with regards to the space needed. We have cloned 1265 projects which occupied a disk space of little more than 43GB. To facilitate the Git repository operations (cloning, reading the commit history) we are using the LibGit2Sharp[1] library.

Table II shows the metrics we are collecting from the software repository together with the information on to how they are acquired. The software process metrics we use were inspired by Madeyski and Jureczko [11] who found that some process metrics (namely NDC and NML) can significantly improve software defect prediction models based on product metrics. Together with these metrics our dataset contains also:

1) Project name
2) File path
3) Commit hash
4) Build commit hash
5) Export date (time stamp of the moment the data were collected)

TABLE II.   Metric harvested from the software repository

Metric	Abbr.	How acquired
NumberOfRevisions	NR	Count the revisions participating in build
NumberOfDistinctCommiters	NDC	Count unique developers involved in revisions in build
NumberOfModifiedLines	NML	Count modified lines since the last build
NumberOfRevisions	NR	Count all the revisions of a given file
BuildDateTimeLocal	BDTL	Build server local date and time of the start of the build
BuildCommitDateTimeLocal	BCDTL	Local time stamp of the commit that caused the build
LastBuildStatus	LBS	Status of the previous build
AuthorIdentification	AI	Author Git user.name setting

The data are stored in a simple Microsoft SQL Server database. The part of the database that stores the metrics

---

[1]https://github.com/libgit2/libgit2sharp

(table Metrics) is presented in Table III. The rest of our database containing tables for defect prediction requests and classification models is not presented here as it is beyond the scope of this paper.

TABLE III. MICROSOFT SQL SERVER DATA STORAGE COLUMN INFORMATION FOR METRICS TABLE

Column name	Type	Nullable
Id	bigint	no
Path	nvarchar(500)	yes
OldPath	nvarchar(500)	yes
NumberOfRevisions	int	yes
NumberOfDistinctCommitters	int	yes
NumberOfModifiedLines	int	yes
BuildResult	int	no
Commit	nvarchar(255)	yes
BuildCommit	nvarchar(255)	yes
ExportDateUtc	datetime	no
NumberOfRevisions	int	yes
BuildDateTimeLocal	datetime	no
BuildCommitDateTimeLocal	datetime	no
BuildProjectName	nvarchar(255)	yes
Author	nvarchar(255)	yes
PreviousBuildResult	int	yes
ProjectName	nvarchar(255)	yes

III. DESCRIPTION OF DATASET

At the time of writing this article we collected the dataset including 11,464,816 rows (representing files that participated in a CI build) in the Metrics database table. The metrics were collected for the 1265 projects gathered in the TravisTorrent database. The count of rows for different project vary drastically (with the minimum at 1, maximum at 2706617, 1st Quartile at 405, 3rd Quartile 2876, median at 938 and mean at 9063). Figure 1 shows the histogram of the rows count over the projects on a logarithmic scale. We have 30,022 distinct commit authors in our database. Interestingly there is a large number of projects with rather small number of data rows in our database, e.g., we have 657 projects with less than 1000 data records (1e+03 mark on Figure 1), meaning all the commits in those projects changed less than 1000 files.
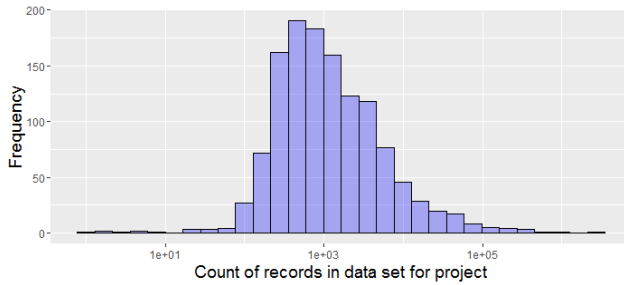

Fig. 1. Log10 scaled histogram of the count of file records per project in our database

We have done some exploratory data analysis for 10 largest projects in our database (the ones with the most rows) and for 10 randomly chosen (using the following SQL command: `select top 10 ... order by newid()` projects with rows count greater than the mean in our database.

TABLE IV. SUMMARY DATA FOR METRICS FOR 10 RANDOM PROJECTS

	NR		NDC		NML		BCDTLl	
	Rand	Top	Rand	Top	Rand	Top	Rand	Top
Min.	1	1	1	1	0	0	0	0
1stQu.	1	1	1	1	1	4	8.011	10.9
Med.	1	1	1	1	4	20	13.01	16.7
Mean	1.95	1.14	1.18	1.04	38.78	89.15	12.41	15.03
3rdQu.	1	1	1	1	26	68	18	20.03
Max.	144	93	18	14	33718	98155	23.02	23.98

Figure 2 includes the names of the selected projects, while Table IV presents the summary data for some of the metrics in those projects.

During our CDP research, we work with both open source and closed source repositories. We share here only the open source based part of our dataset which poses a threat as it may not generalize to other contexts, e.g., commercial/closed source software projects. It is also worth mentioning that the TravisTorrent dataset we build upon restricted the project space using filtering criteria to Ruby or Java non-fork, non-toy, somewhat popular (> 10 watchers on GITHUB) projects with some history of TRAVIS CI use (> 50 builds) [7].

The goal of the paper is to briefly describe the dataset we have shared. However, our role as researchers, even if generally beyond the scope of this data paper, is not only to collect data, but also to transform them into understanding. As an example of interesting insights or ideas for what future research questions could be answered with the provided dataset, we are presenting the box plots for the build commit times in the 10 largest large and 10 randomly chosen projects. Assuming these small samples are representative to some extent, we could draw two interesting, albeit preliminary, hypotheses from this plot. The first one, derived from Figure 2 and Table IV, is that for the larger projects commits that trigger the CI build are done later in day time, whereas they are done earlier in the average. The second one is that the open source projects integrations are done generally well before 8 PM so on average open source developers are not night owls as they are usually perceived. Actually, the builds are done based on commits done in normal business hours, between 9 AM and 5 PM.

IV. FUTURE WORK AND CONCLUSIONS

We are using the dataset presented in this paper to create prediction models for continuous prediction of success/fail continuous integrations. We are working currently on a pilot CDP project in a commercial software development environment, at a company managed by one of the authors of this paper, but we are going to release the whole CDP project together with tools used to gather the data using dual open source and commercial licences. We are going to enrich the dataset we collected with more metrics that, according to existing empirical evidence by other researchers, can be used as features in our prediction models. It might even be possible (through cooperation with one of the cloud providers) to expose our prediction models over the web in a ready to use manner to aid development of open source projects.

By opening our database to the public we hope to attract an audience and feedback to our project, as well as attract researchers to enhance the dataset by new metrics or new
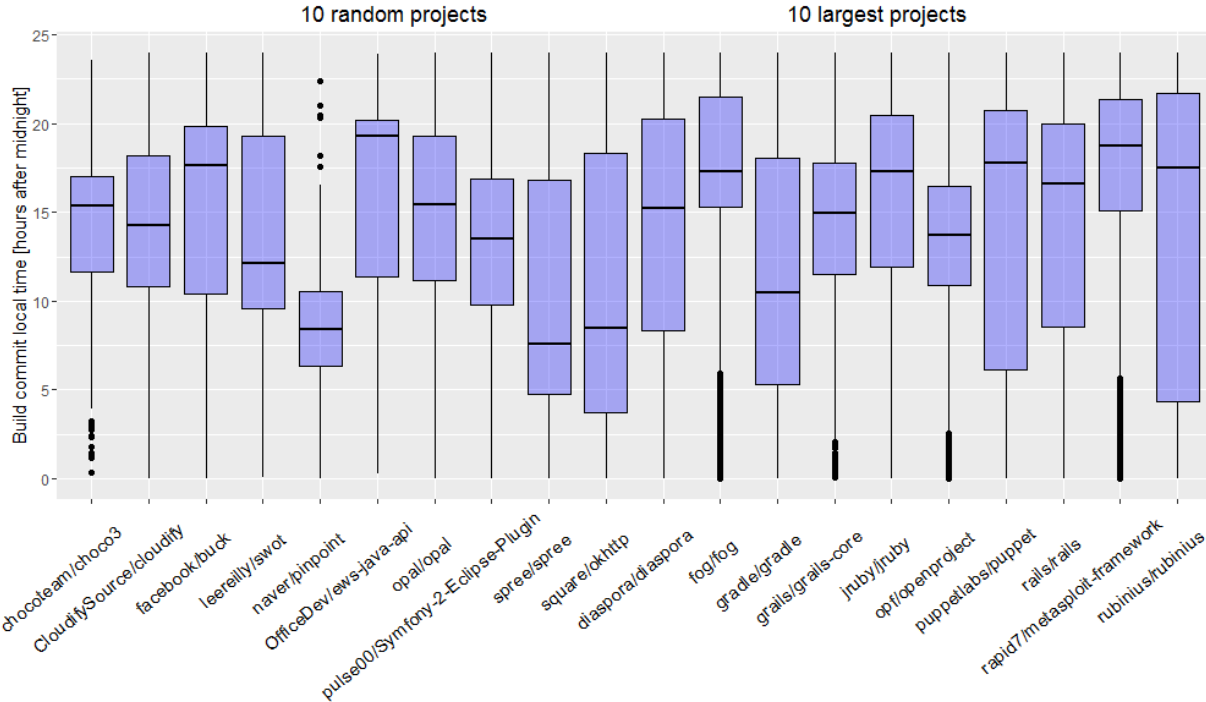
Fig. 2. Build commit local time for 10 random and 10 largest projects in our dataset

kinds of metrics, to build prediction models on the basis of a dataset from more than one thousand of software projects and even more unique developers, or to predict other dependent variables that can be useful for practitioners. Areas in which our data could aid future research are broad and include: defect prediction on software change level, stability and maturity studies on long running software projects, developers activity and results examination, or exploration of trends in continuous integration over a period of time.

The database is available as a CSV (with semicolon as separator and double quote as string delimiter) file at figshare[2] and as a Microsoft SQL Server 2012 dump file[3].

ACKNOWLEDGMENT

The authors would like to thank the authors of TravisTorrent [7] who provided us a set of data that we were able to extend to use for our purposes. We would also like to thank the employees of the CODEFUSION company for their valuable input and help in tools development.

REFERENCES

[1] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, vol. 14, no. 5, pp. 540–578, 2009.

[2] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Software Engineering*, vol. 17, pp. 531–577, 2012.

[3] M. Jureczko and L. Madeyski, "Cross–Project Defect Prediction With Respect To Code Ownership Model: An Empirical Study," *e-Informatica Software Engineering Journal*, vol. 9, no. 1, pp. 21–35, 2015, DOI: 10.5277/e-Inf150102.

[4] S. Kim, E. J. Whitehead, Jr., and Y. Zhang, "Classifying software changes: Clean or buggy?" *IEEE Trans. Softw. Eng.*, vol. 34, no. 2, pp. 181–196, Mar. 2008, DOI: 10.1109/TSE.2007.70773.

[5] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A large-scale empirical study of just-in-time quality assurance," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 757–773, 2013.

[6] X. Yang, D. Lo, X. Xia, Y. Zhang, and J. Sun, "Deep learning for just-in-time defect prediction," in *Software Quality, Reliability and Security (QRS), 2015 IEEE International Conference on*, Aug 2015, pp. 17–26.

[7] M. Beller, G. Gousios, and A. Zaidman, "TravisTorrent: Synthesizing Travis CI and GitHub for Full-Stack Research on Continuous Integration," in *Proceedings of the 14th Working onference on Mining Software Repositories*, 2017.

[8] L. Madeyski and M. Kawalerowicz, "Continuous Test-Driven Development—A Novel Agile Software Development Practice and Supporting Tool," in *ENASE 2013 - Proceedings of the 8th International Conference on Evaluation of Novel Approaches to Software Engineering*, J. Filipe and L. Maciaszek, Eds., 2013, pp. 260–267.

[9] L. Madeyski and M. Kawalerowicz, "Software Engineering Needs Agile Experimentation: A New Practice and Supporting Tool," in *Software Engineering: Challenges and Solutions*, ser. Advances in Intelligent Systems and Computing, L. Madeyski, M. Śmiałek, B. Hnatkowska, and Z. Huzar, Eds. Springer, 2017, vol. 504, pp. 149–162.

[10] G. Gousios, "The GHTorrent dataset and tool suite," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 233–236.

[11] L. Madeyski and M. Jureczko, "Which process metrics can significantly improve defect prediction models? An empirical study," *Software Quality Journal*, vol. 23, no. 3, pp. 393–422, 2015, DOI: 10.1007/s11219-014-9241-7.

[2]https://figshare.com/s/302814fa28cb1fbde705
[3]https://figshare.com/s/394e2f8d7dc6da405721