Exploit Those Code Reviews! Bigger Data for Deeper Learning

Robert Heumüller
Otto-von-Guericke University
Magdeburg, Germany
robert.heumueller@ovgu.de

Sebastian Nielebock
Otto-von-Guericke University
Magdeburg, Germany
sebastian.nielebock@ovgu.de

Frank Ortmeier
Otto-von-Guericke University
Magdeburg, Germany
frank.ortmeier@ovgu.de

ABSTRACT

Modern code review (MCR) processes are prevalent in most organizations that develop software due to benefits in quality assurance and knowledge transfer. With the rise of collaborative software development platforms like GitHub and Bitbucket, today, millions of projects share not only their code but also their review data. Although researchers have tried to exploit this data for more than a decade, most of that knowledge remains a buried treasure. A crucial catalyst for many advances in deep learning, however, is the accessibility of large-scale standard datasets for different learning tasks. This paper presents the ETCR (Exploit Those Code Reviews!) infrastructure for mining MCR datasets from any GitHub project practicing pull-request-based development. We demonstrate its effectiveness with ETCR-Elasticsearch, a dataset of >231k review comments for >47k Java file revisions in >40k pull-requests from the Elasticsearch project. ETCR is designed with the challenge of deep learning in mind. Compared to previous datasets, ETCR datasets include all information for linking review comments to nodes in the respective program's Abstract Syntax Tree.

CCS CONCEPTS

• **Computing methodologies** → *Machine learning*; • **General and reference** → *Experimentation*; • **Software and its engineering** → *Software development techniques*; *Collaboration in software development*.

KEYWORDS

datasets, code review, source code, deep learning

ACM Reference Format:
Robert Heumüller, Sebastian Nielebock, and Frank Ortmeier. 2021. Exploit Those Code Reviews! Bigger Data for Deeper Learning. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21), August 23–28, 2021, Athens, Greece.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3468264.3473110

1 INTRODUCTION

Modern Code Review (MCR) processes are state of the art in most organizations that develop software, including open-source projects and global players like Microsoft or Google [1, 15]. While there is a consensus that MCR pays for itself through benefits in quality assurance and knowledge transfer in practice, empirical data on time invested into MCR is scarce. Case studies at Google and for 45 open-source projects found that developers invested on average 3.2h to 6.4h per week [2, 15] in MCR. Though the generality of these studies may be limited, the numbers indicate that the time invests are very significant. Consequently, much can be gained by optimizing MCR processes' efficiency (i.e., reducing the time per review) and effectiveness (i.e., ensuring developers' review time is spent where it matters), for example, on those sections of a new feature that are the most complex or error-prone. As current research demonstrates, the advances in deep learning enable researchers to build models capable of exploiting review data to learn code changes or share reviews between similar source code in different projects [6, 17]. However, deep learning requires datasets of sufficient quantity and quality. In domains such as computer vision, standard datasets have proved to be a crucial catalyst for many significant advances. With ETCR we aim to establish a standard infrastructure for creating MCR datasets geared towards deep learning for automating code review activities. To demonstrate its effectiveness, with ETCR-Elasticsearch, we publish a first dataset for the Elasticsearch project, which includes 231k review comments, 91k of which addressed Java source code that is available in the dataset. As an ETCR dataset, it provides everything required for linking between review comments and specific Abstract Syntax Tree (AST) nodes. Compared to previous code review datasets [5, 13, 14, 17, 18, 20], ETCR greatly facilitates the creation of further datasets and giving access to potentially millions of projects by using GitHub pull requests[1]. We encourage fellow researchers to explore ETCR-Elasticsearch for their work and to reuse ETCR to create other datasets. Both artifacts are available on Zenodo [9, 10] and the accompanying screencasts can be found on YouTube[2].

2 ETCR

In this section, we give an overview of ETCR. Particularly, we provide insights into ETCR's data model and its general data collection process in Section 2.1. Then, we describe ETCR-Elasticsearch, the first sample dataset generated by ETCR (cf. Section 2.2).

2.1 Data Model and Collection

We constructed ETCR to collect as much data as possible from an ordinary MCR process to make it applicable for diverse deep learning applications. Thus, ETCR extracts data from all the individual steps of MCR processes as described by previous work [8, 15]. Concretely, this encompasses (1) the creation, (2) preview, and (3) commenting of code changes as well as (4) addressing those comments, and

[1] Source: GitHub Blog, Visited May 5th 2021
[2] ETCR Overview Screencast, ETCR Tutorial Screencast

ESEC/FSE '21, August 23–28, 2021, Athens, Greece

ACM ISBN 978-1-4503-8562-6/21/08.
https://doi.org/10.1145/3468264.3473110

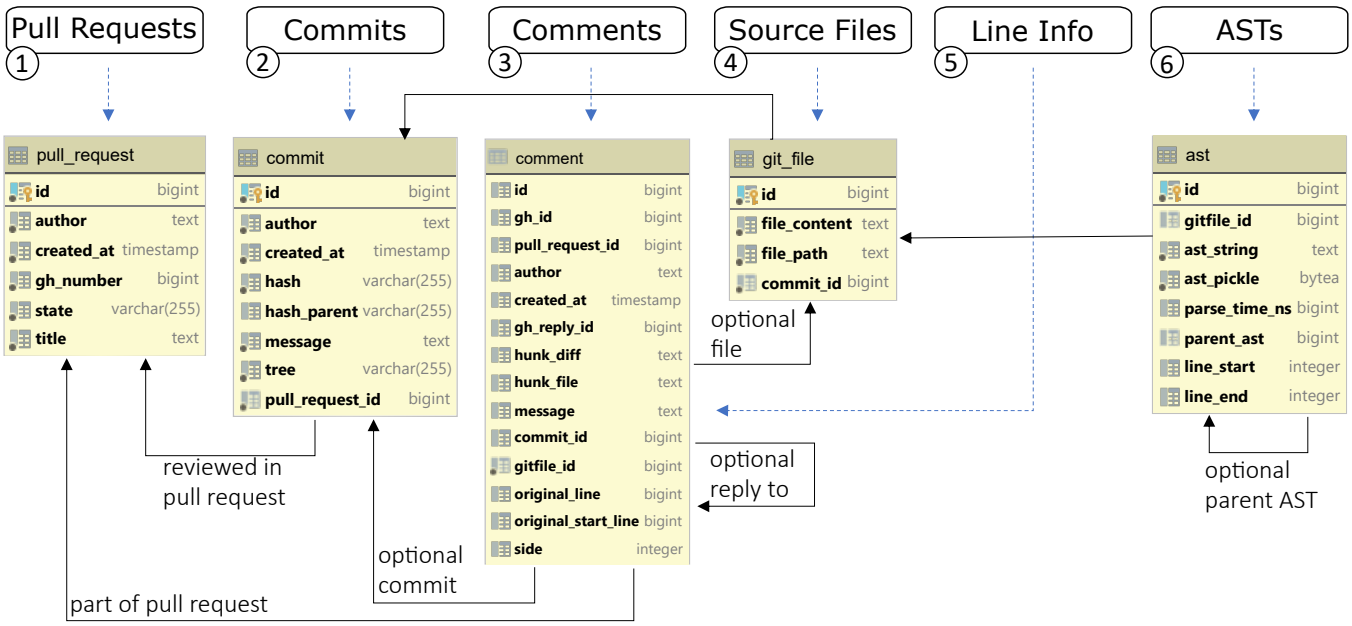

Figure 1: ETCR Data Collection Process and Relational Datamodel

eventually (5) approving the final changes. Note that steps (2)-(4) typically can take multiple iterations. In deciding which data to include, we took into account recent MCR-related deep learning applications to identify additional valuable information [17].

Figure 1 illustrates the six steps and corresponding data model relations that ETCR uses to download, process, and store data in a relational database. Its data model is an extension of previous code review data models [13, 18]. Steps ①-④ use ETCR's main, Kotlin-based web crawler[3] to download the review data and source code from GitHub via its HTTP-API[4]. We used this API as the source for code reviews since it does not alter the project histories like Gerrit does, as identified by previous work [13, 14]. Steps ⑤ and ⑥ use Python modules to augment the resulting dataset. The minimal configuration requires only the target repository identifier (e.g., `elastic/elasticsearch`), a valid GitHub API key, and the connection details for a PostgreSQL database. In the following, we only explain the most important aspects of the data model. Further documentation is included in the artifact [10].

Step ① downloads the meta-data (i.e., review identifier, author, topic, and open or closed state) for the specified project. Step ② uses this data to retrieve all commits (i.e., initial and addressed code changes) for each pull request, storing their meta-data and messages. Since a commit can be featured in more than one pull request, except for the `id`, it is possible to have duplicate entries in the commit table[5]. When multiple commits are associated with one pull request, each following commit typically addresses previous comments and initiates a new review iteration. Step ③ downloads the review comments related to each pull request. In GitHub, one can comment on pull requests as a whole or particular code changes. In the first case, there is no direct reference to the code or even a specific commit. Thus, we store only the message and author information. If desired, one can use the timestamps to indirectly associate those messages to commits. In the second case, we also store the concrete change information, including a reference to the respective commit, the path to the changed file, the change hunk within that file, and a link to that file revision (cf. ④). Note that a commented line may refer to a commit that is not part of the pull request, as reviewers can also comment on the context of changed code. Finally, comments may also be replies to other comments, depicted using self-references. Step ④ downloads the correct revisions of any file addressed by a comment. A change always involves two revisions, *before* and *after*. We only include the *after revisions* in the data model because the *before revisions* can be restored by rewinding the change hunks from step ③. Step ⑤ once more uses the GitHub API to supplement the comments table with the line information (i.e., the commented single lines or line ranges), which we found beneficial from other datasets [13, 14]. The *side* attribute identifies whether lines were added or deleted[6]. Step ⑥ uses *javalang*[7] to parse the java file revisions and stores the serialized file- and method-level ASTs. Links to their respective parents distinguish method-level ASTs from file-level ASTs. Stored ASTs can be deserialized and processed using *javalang*. To facilitate linking from comments to method ASTs, we also include the methods' line ranges.

Next, we introduce ETCR-Elasticsearch, the dataset created by applying the above process to the GitHub repository of the *elasticsearch* search engine. Note that for data protection we pseudonymized all author names of pull requests, commits, and comments using sha-224 hashes.

[3] We credit Tien Dô Nam for the implementation.
[4] https://docs.github.com/en/rest/reference/pulls
[5] for ETCR-Elasticsearch this happens for ≈ 8% of the commits
[6] left side (=0) means addition while right side (=1) relates to deletion
[7] https://github.com/c2nes/javalang

Table 1: Descriptive Statistics for ETCR-Elasticsearch

Basic Statistic	Value
# Pull Requests	40,323
# Contributors	3,816
# Commits	154,844
# Distinct Files (Java Files)	15,211 (12,492)
# File Revisions (Java Files)	58,078 (47,014)
# Comments (% Replies)	231,627 (16%)

Derived Statistic	Value
# PRs per Author (Median) (0.9 Percentile)	15.6 (1) (5)
# Commits per Author (Median) (0.9 Percentile)	66.6 (3) (18,0)
# Comments per Author (Median) (0.9 Percentile)	82.15 (2) (16,5)
# Comments to Java Files	97,831
# Java File ASTs (% of Java File Revisions)	45,431 (97%)
# Java Method ASTs (Unique)	1.09M (228k)
# Comments to Java Files with full Line Information and ASTs (% of Java File Revisions)	**91,249 (93%)**

2.2 ETCR-Elasticsearch

We generated the ETCR-Elasticsearch dataset in Dec. 2020 by applying ETCR to the GitHub repository of the *elasticsearch* search engine. This took roughly a week, primarily due to GitHub's API rate limits. We selected *elasticsearch* because, besides having a compatible license, it is a very established Java project, first released in 2010. Since then, ca. 3.8k contributors submitted more than 40k pull requests and ca. 155k distinct commits. Next, we discuss ETCR-Elasticsearch's benefits regarding its quantity and quality.

Quantity. To judge the quantity of ETCR-Elasticsearch specifically concerning deep learning applications, we compare it to similar existing datasets. Moreover, we describe ETCR-Elasticsearch through a set of descriptive statistics in Table 1.

As a first reference, we selected two recent code change and review datasets that were successfully evaluated for training neural machine translation (NMT) architectures. Tufano et al. created a dataset for learning code changes consisting of ca. 240k method pairs (i.e., 480k method revisions) from ca. 80k pull requests. Each method pair consists of the method's source code before and after a completed pull request [16]. Their current research also includes review comments, forming so-called triplets. The corresponding dataset contains ca. 17k triplets [17]. Therefore, with 1.09M method revisions, 228k of which are unique, ETCR-Elasticsearch is comparable to this larger dataset. Compared to the triplet-dataset, ETCR-Elasticsearch contains 2.6 times as many comments with fully available source code (91k). Table 3 shows that the number of comments of ETCR-Elasticsearch is comparable to other datasets, particularly since the larger datasets used multiple projects (i.e., Paixao et al. [14], Yang et al. [18]).

As a second comparison, we chose two representative datasets from text classification, namely, IMDB-Reviews (50k instances, 2 classes) for sentiment analysis and News-Categories (200k instances, 41 classes) for topic classification [11, 12]. Both learning tasks have analogs in classifying review comments or code. Quantitatively, ETCR-Elasticsearch features pull requests, commits, file revisions, and comments in the same magnitude (i.e., $10^4 - 10^5$). We expect the number of classes to range similarly from 10-100 for many learning tasks (e.g., classifying topics in review comments).

Table 2: Exploring Comments in ETCR-Elasticsearch

Refactoring Topic Queries	Count
'%extract %constant%'	17
'%extract %variable%'	14
'%magic value%'	9
'%magic number%'	22
'%extract %method%'	106
'%separate method%'	76
'%separate class%'	70
'%extract %class%'	37
'%a factory%'	23
'%singleton%'	52

Actionability Queries	Count
'%make this a %'	228
'%can do better%'	23
'%should %refactor%'	174
'%should refactor%'	15
'%this should be%'	931
'%we should%'	9,192
'%you should%'	486
'%move this to%'	149

Thus, we conclude that the quantity of data in ETCR-Elasticsearch is very likely sufficient for many deep learning applications. Moreover, if required, researchers can use the ETCR infrastructure to easily create other datasets.

Quality. In the following, we discuss the dataset's quality by considering its practical *application* and *diversity*.

We see ETCR's main *application* in facilitating research on the automation of code review using deep learning. Besides Tufano et al.'s work mentioned above, Guo et al. developed rsharer, which uses deep learning to share reviews between code clones in different projects. These recent approaches exploit MCR data to assist at the method- or code-snippet level. We expect future approaches to use MCR data at a finer granularity. For example, emerging techniques could learn to identify and fix flaws regarding particular AST nodes, e.g., expressions, method invocations, or conditions. To the best of our knowledge, researchers did not yet attempt this. Thus, we designed ETCR to provide everything required for developing such models: complete review comments and source code of the addressed revisions, ASTs at the file- and method level, and utilities for linking comments to AST nodes.

We further demonstrate ETCR's *application* in an explorative look at its suitability to the practical learning task of semantic classification. We consider two meta-classes: ⓐ classifying the refactoring-topics and ⓑ the actionability of review comments (i.e., whether comments trigger changes). Systems based on these classifications could inform developers about specific issues in their code (ⓐ) or give reviewers feedback on how likely their remarks will lead to changes rather than further discussion (ⓑ). Such classifiers could benefit from ETCR's combination of comments, code changes, and AST nodes in multiple ways. As a first example, context information from the addressed source code could enable finer discrimination of review topics than, e.g., binary classifications from earlier work (ⓐ) [19]. From this, we derive a new research question, particularly for ETCR datasets: *Can we build classifiers with superior performance by combining comment text classification with a context representation derived from the corresponding source code?* As a second example, based on the dataset, one can tell whether code sections changed shortly after being commented on by a reviewer. This knowledge could be used as ground truth for actionability classifiers, as it suggests that the comment is likely to have induced the change (ⓑ).

Table 2 further shows the *diversity* of ETCR-Elasticsearch by using simple textual queries to count the appearances of comments referring to the metaclasses introduced above. The first four queries

Table 3: Comparison of ETCR-Elasticsearch to existing code review datasets.

Dataset	Comments (Number)	Line Data	Source Code	ASTs	Applicability
Tufano et al. [17]	Yes (17, 194[8])	-	Abstracted Methods	-	GitHub, Gerrit
Paixao et al. [14]	Yes (507, 268)	Yes	Revision-Snapshots	-	Specific Gerrit Proj.
Mukadam et al. [13]	Yes (106, 439)	Yes	Metadata	-	Gerrit
Yang et al. [18]	Yes (6, 375, 128[9])	-	Metadata	-	Gerrit
Gousios and Zaidman [5]	Metadata (-)	-	-	-	GHTorrent
Zhang et al. [20]	Metadata (-)	-	-	-	GHTorrent
ETCR-Elasticsearch	Yes (231, 627)	Yes	Revision-Snapshots	File&Method	GitHub

refer to magic-value anti-patterns [3]. The second four address design flaws that developers could remedy by extracting methods or classes. [3]. The final queries search for particular design patterns: factory and singleton [4].

Regarding actionability, we defined similar queries. Here too, the simple indicators worked surprisingly well. Interestingly, the query "%we should%" by far dominates, giving a positive insight into the tone of the conversation.

The first author validated ten random comments per query to gauge the number of false positives. For both types of indicators, the average precision in the random sample was 0.78, with only one query scoring below 0.5. This investigation shows that all of the exemplary topics frequently appear in ETCR-Elasticsearch, underlining the diversity of the data.

Finally, we examine the per-authors statistics[10] in Table 1 as another indicator of diversity. As the significant differences between the averages and the medians or 0.9 percentiles indicate, a small number of people make the vast majority of all contributions. Further investigating this, 90% of all commits and comments are accounted for by ca. 90 developers. Thus, we conclude that the ETCR-Elasticsearch reflects the diverse styles, experiences, and priorities of many developers.

3 RELATED DATASETS

This section gives an overview of the most relevant existing, MCR-related datasets and their intended research goals. Table 3 shows a direct comparison of these datasets to ETCR datasets and in particular to ETCR-Elasticsearch.

Tufano et al. created two datasets for predicting method-level code changes via deep learning. The first one only used code changes from 80k pull requests, whereas the more recent one also incorporates review comments (based on ca. 17k changes) from GitHub and Gerrit [16, 17]. Compared to ETCR the code is abstracted, and their selection criteria drastically reduce the number of retained instances. Thus, the dataset may be less suited for reuse in other research avenues. The Code Review Open Platform (CROP) by Paixao et al. [14] is a dataset based on specific Gerrit projects consisting of ca. 507k comments. Since Gerrit is known to rewrite repository histories [13, 14], CROP only selects projects that keep copies of their original history in separate repositories. Therefore CROP offers full snapshots of all relevant revisions. This project selection, however, limits its extensibility compared to ETCR-datasets.

The dataset from Mukadam et al. [13] suffers from similar issues and lacks actual source code. Even though it also provides line numbers, it has fewer comments in total (ca. 106k vs. 231k, cf. Table 3) and provides only source code metadata. Yang et al. [18] have the largest number of comments (ca. 6M). However, they also rely on Gerrit, lack line information, and provide only source code metadata, since they focus on the people, process, and product aspects of MCR. An earlier, no longer available dataset had similar limitations [7]. Finally, Gousios and Zaidman [5] and Zhang et al. [20] created datasets of GitHub pull requests that contain mostly metadata (e.g., comment counts, etc.) for empirical research on pull-based development.

In summary, compared to existing datasets, ETCR datasets contain everything required to link review comments to complete source code. To the best of our knowledge, ETCR is the first to offer ASTs. Moreover, building on GitHub, ETCR is more extensible compared to Gerrit-based approaches.

4 CONCLUSION

In this paper, we presented the ETCR infrastructure for creating MCR datasets by mining GitHub pull requests. Using the example of ETCR-Elasticsearch, we showed the richness of such datasets in terms of quantity and quality. We compared ETCR datasets to existing MCR datasets, showing how we combine their strengths to provide a foundation for standard datasets in code review automation. In our future work, we plan to (a) release further datasets, (b) to create adapters for deep learning frameworks such as PyTorch, (c) research semantic review classification that combines comment- and source code-features, and (d) further advance the field of deep learning for code review automation.

[8] retained after filtering an initial set of 231, 439 reviewers' comments
[9] based on updated data from kin-y.github.io/miningReviewRepo/
[10] Note that we excluded 27,407 bot-comments for these statistics.

REFERENCES

[1] Alberto Bacchelli and Christian Bird. 2013. Expectations, Outcomes, and Challenges of Modern Code Review. In *Proceedings of the 35th International Conference on Software Engineering (ICSE)*. IEEE, 712–721. https://doi.org/10.1109/ICSE.2013.6606617

[2] Amiangshu Bosu and Jeffrey C. Carver. 2013. Impact of Peer Code Review on Peer Impression Formation: A Survey. In *Proceedings of the 7th International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 133–142. https://doi.org/10.1109/ESEM.2013.23

[3] Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. 1999. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman Publishing Co., Inc., USA. ISBN-13: 978-0-201-48567-7.

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman Publishing Co., Inc., USA. ISBN-13: 978-0-201-63361-0.

[5] Georgios Gousios and Andy Zaidman. 2014. A dataset for pull-based development research. In *Proceedings of the 11th International Workshop on Mining Software Repositories (MSR)*. ACM, 368–371. https://doi.org/10.1145/2597073.2597122

[6] Chenkai Guo, Dengrong Huang, Naipeng Dong, Quanqi Ye, Jing Xu, Yaqing Fan, Hui Yang, and Yifan Xu. 2019. Deep Review Sharing. In *Proceedings of the*

26th International Conference on Software Analysis, Evolution, and Reengineering (SANER). IEEE, 61–72. https://doi.org/10.1109/SANER.2019.8668037

[7] Kazuki Hamasaki, Raula Gaikovina Kula, Norihiro Yoshida, A. E. Camargo Cruz, Kenji Fujiwara, and Hajimu Iida. 2013. Who does what during a code review? Datasets of OSS peer review repositories. In *Proceedings of the 10th International Workshop on Mining Software Repositories (MSR)*. IEEE. https://doi.org/10.1109/msr.2013.6624003

[8] Robert Heumüller. 2021. Learning to Boost the Efficiency of Modern Code Review. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. 275–277. https://doi.org/10.1109/ICSE-Companion52605.2021.00126

[9] Robert Heumüller, Sebastian Nielebock, and Frank Ortmeier. 2021. ETCR-Elasticsearch Dataset. https://doi.org/10.5281/ZENODO.5079076

[10] Robert Heumüller, Sebastian Nielebock, and Frank Ortmeier. 2021. ETCR Infrastructure. https://doi.org/10.5281/ZENODO.4739592

[11] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meetings of the Association for Computational Linguistics (ACL) Human Language Technologies*. ACL, 142–150. http://www.aclweb.org/anthology/P11-1015

[12] Rishabh Misra. 2018. *News Category Dataset*. https://doi.org/10.13140/RG.2.2.20331.18729

[13] Murtuza Mukadam, Christian Bird, and Peter C. Rigby. 2013. Gerrit software code review data from Android. In *Proceedings of the 10th International Workshop on Mining Software Repositories (MSR)*. IEEE, 45–48. https://doi.org/10.1109/MSR.2013.6624002

[14] Matheus Paixao, Jens Krinke, Donggyun Han, and Mark Harman. 2018. CROP: Linking Code Reviews to Source Code Changes. In *Proceedings of the 15th International Workshop on Mining Software Repositories (MSR)*. ACM, 46–49. https://doi.org/10.1145/3196398.3196466

[15] Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern Code Review: A Case Study at Google. In *Proceedings of the 40th International Conference on Software Engineering (ICSE) Software Engineering in Practice Track*. ACM, 181–190. https://doi.org/10.1145/3183519.3183525

[16] Michele Tufano, Jevgenija Pantiuchina, Cody Watson, Gabriele Bavota, and Denys Poshyvanyk. 2019. On Learning Meaningful Code Changes via Neural Machine Translation. In *Proceedings of the 41st International Conference on Software Engineering (ICSE)*. IEEE, 25–36. https://doi.org/10.1109/ICSE.2019.00021

[17] Rosalia Tufano, Luca Pascarella, Michele Tufano, Denys Poshyvanyk, and Gabriele Bavota. 2021. Towards Automating Code Review Activities. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE. https://doi.org/10.1109/icse43902.2021.00027

[18] Xin Yang, Raula Gaikovina Kula, Norihiro Yoshida, and Hajimu Iida. 2016. Mining the modern code review repositories: a dataset of people, process and product. In *Proceedings of the 13th International Workshop on Mining Software Repositories (MSR)*. ACM, 460–463. https://doi.org/10.1145/2901739.2903504

[19] Farida El Zanaty, Toshiki Hirao, Shane McIntosh, Akinori Ihara, and Kenichi Matsumoto. 2018. An empirical study of design discussions in code review. In *Proceedings of the 12th International Symposium on Empirical Software Engineering and Measurement (ESEM)*. ACM. https://doi.org/10.1145/3239235.3239525

[20] Xunhui Zhang, Ayushi Rastogi, and Yue Yu. 2020. On the Shoulders of Giants: A New Dataset for Pull-based Development Research. In *Proceedings of the 17th International Workshop on Mining Software Repositories (MSR)*. ACM, 543–547. https://doi.org/10.1145/3379597.3387489