Code Reviewer Recommendation for High Risk Diffs at Scale: Workflow, Recommender, and Live Experiments

Aishwarya Girish Paraspatki
Brandon Reznicek
aishp@meta.com
reznicek@meta.com
Meta Platforms, Inc.
Menlo Park, CA, USA

Rui Abreu
Ford Garberson
Audris Mockus*
ruiabreu@meta.com
fgarberson@meta.com
audris@meta.com
Meta Platforms, Inc.
Menlo Park, CA, USA

Nachiappan Nagappan
Peter C. Rigby†
nnachi@meta.com
pcr@meta.com
Meta Platforms, Inc.
Menlo Park, CA, USA

Abstract

Code review is an important gate for code changes, but review effort and expertise are limited. Defect prediction and just-in-time risk modeling aim to identify changes that are most likely to introduce defects so that teams can focus additional attention where it is most useful. We study whether risk-aware interventions, targeted specifically at these high-risk changes, can improve review outcomes while preserving developer velocity. In our setting, *risk* refers to the likelihood that a code change will introduce a defect after landing. We quantify this defect risk with the Diff Risk Score (DRS) used to identify high-risk diffs and investigate three research questions related to code review: (RQ1) whether a dedicated workflow improves review behavior on high-risk diffs, (RQ2) whether we can design a reviewer recommender tailored to risky diffs, and (RQ3) how well that new recommender performs in an A/B test across thousands of high risk diffs.

For RQ1, we introduce the Risky Diff Reviewer (RDR) workflow, which assigns an additional recommended reviewer for high-risk diffs and shows a bypassable pre-land dialog when authors attempt to land without resolving the risk signal. In a randomized trial on the top 5% of diffs by diff defect risk (over 15k diffs), RDR increases review depth and collaboration and increases the fraction of diffs whose final version falls below the high-risk threshold by 9.56%, with a minor 0.45% increase in diff reviewing time.

For RQ2, we design a new reviewer recommender (RecRDR) for risky diffs. RecRDR uses a continuous, weighted ground truth that emphasizes file experience, author collaboration, interaction quality, and recency. In offline backtests, RecRDR improves Top-1 and Top-3 action rates over the baseline recommender.

For RQ3, we evaluate RecRDR using an A/B test within the RDR workflow on over 5k diffs. RecRDR increases engagement from the recommended reviewer, with 9.94% more comments and 7.77% more acceptances, and we do not observe statistically significant regressions in guardrail metrics such as diff reviewing and processing time.

CCS Concepts

• **Software and its engineering → Software organization and properties**.

Keywords

Code Review, Defect Modelling, Developer Productivity

ACM Reference Format:
Aishwarya Girish Paraspatki, Brandon Reznicek, Rui Abreu, Ford Garberson, Audris Mockus, Nachiappan Nagappan, and Peter C. Rigby. 2026. Code Reviewer Recommendation for High Risk Diffs at Scale: Workflow, Recommender, and Live Experiments. In *34th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE Companion '26), July 05–09, 2026, Montreal, QC, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3803437.3805220

1 Introduction

We aim to improve the quality of code review for the riskiest changes, *i.e.* those that might introduce a defect, without harming developer velocity. Defect prediction models have existed for decades [25, 32] and some organizations surface risk signals to engineers during review [66]. A common mitigation is to suggest extra reviewers for risky changes. Despite this long history, we are unaware of prior industry studies that rigorously test whether risk-aware reviewer assignment and workflow interventions actually improve review outcomes in a large organization.

We operate in a large, fast-moving environment where engineers push code frequently and peer review is the primary gate before landing. We surface the Diff Risk Score (DRS) directly in the review UI for high-risk diffs, highlight risky snippets, and provide a soft-blocking warning that encourages mitigation [2]. Building on this signal, we introduce an integrated experience that automatically assigns a top reviewer for risky diffs and prompts authors for justification when attempting to land without resolving the risk. We also redesign reviewer recommendation specifically for risky diffs, emphasizing recent, meaningful engagement and collaboration over historical, latency-oriented signals.

We address three needs. First, we need to quantify whether a dedicated workflow for risky diffs drives more thorough and expert review. Second, we need a recommender that identifies contextually relevant reviewers for risky changes, not merely people who have

*Mockus is also a professor at University of Tennessee, Knoxville, USA.
†Rigby is also a professor at Concordia University in Montreal, QC, Canada.

FSE Companion '26, Montreal, QC, Canada

ACM ISBN 979-8-4007-2636-1/2026/07
https://doi.org/10.1145/3803437.3805220

reviewed in the past. Third, we need to validate these interventions in live systems using Randomized Controlled Trials (RCT) with goal, tracking, and guardrail metrics that are instrumented consistently across reviews. We discuss our three research questions below summarizing the motivation, methodology, and results.

RQ 1. Risky Diff Reviewer (RDR) Experiment

Expt. 1: Can we drive more thorough review by more qualified reviewers on risky diffs?

Motivation. When a diff is flagged as high risk, additional scrutiny can help authors and reviewers surface issues earlier. We study whether a lightweight, risk-aware workflow changes review behavior on the riskiest diffs while preserving velocity. We suggested an additional reviewer in the Risky Diff Reviewer (RDR) workflow using the existing RevRec recommender [63].

Method. We ran an A/B test on over 15k diffs[1] on non-config diffs, targeting the top 5% by Diff Risk Score (DRS). In treatment condition, we automatically assigned a DRS reviewer and showed a bypassable pre-land dialog when authors attempted to land without approval from the DRS reviewer or a top recommended reviewer.

Results summary. The RDR workflow increased review depth and collaboration (more comments and revisions) and increased the fraction of diffs whose final version fell below the high-risk threshold. Reviewer engagement increased, with a low bypass rate indicating that most authors used the workflow as intended. Changes in diff processing time were modest.

RQ 2. A recommender model for risky diffs

Can we more accurately recommend the reviewers for risky diffs?

Motivation. Engineers reported that the baseline reviewer recommender sometimes suggested reviewers who lacked the most relevant context for high-risk diffs. We explore whether a recommender tuned for risky diffs can surface more contextually appropriate reviewers.

Method. We designed, RecRDR, a new recommender model for RDR that emphasizes recent, meaningful engagement using low-latency signals such as recency strength and author-reviewer interaction history, and we trained it using a weighted ground truth rather than a binary "did they review" label.

Results summary. In offline backtests, the best model variant improved Top-1 and Top-3 recommended reviewer action rates over the baseline (RevRec), indicating that the added recency and interaction signals improved recommendation quality.

RQ 3. RevRec vs RecRDR Experiment

How well does the new recommender work in practice?

Motivation. Offline gains may not translate to real reviewer behavior, especially when availability and workload vary. We evaluate whether RecRDR improves engagement from the recommended reviewer under real review conditions.

Method. We ran an A/B test on over 5k high-risk diffs within the RDR workflow, comparing RecRDR to the baseline recommender while keeping the rest of the experience unchanged. We tracked recommended reviewer engagement as goal metrics, along with velocity-focused guardrails.

Results summary. RecRDR increased engagement from recommended reviewers (more comments and acceptances) without observable regressions in guardrail metrics such as diff processing time and review duration. These results suggest that tuning recommendations for recent and relevant expertise improves risky-diff review in practice.

The remainder of this paper is structured as follows. In Section 2, we provide background on software development and code review, and we describe the Diff Risk Score (DRS) model. In Section 3, we present the design of the Risky Diff Reviewer (RDR) workflow and how it integrates a UI-surfaced risk signal with targeted reviewer assignment and a soft-blocking pre-land prompt. In Section 4, we describe how we create a new reviewer recommender for risky diffs, including the revised ground truth and the low-latency features used at recommendation time. In Section 5, we describe our experimental methodology and the Randomized Controlled Trials (RCTs) used to evaluate both the RDR workflow and the new recommender. In Section 6, we report the results for RQ1–RQ3, including offline backtests and A/B tests. In Section 7, we position and discuss our contributions in the research literature. In Section 8, we discuss threats to validity. In Section 9, we provide conclusions and contributions.

2 Background

2.1 Software Development at Meta

Meta develops software for both its servers and client devices, including specialized hardware devices. This approach facilitates swift software updates and provides meticulous control over versioning and configurations. At Meta, this deployment strategy has cultivated a routine of frequently "pushing" new code to production. Prior to any push, the code undergoes peer review, in-house user testing, and comprehensive automated and canary tests. Once deployed, engineers scrutinize logs to spot potential problems.

At Meta, it is customary for engineers to review each other's code. This process serves several functions [71]. Firstly, it motivates the original coder to maintain high coding standards. Secondly, a reviewing engineer, with a fresh perspective, might detect flaws or propose better solutions. Thirdly, it promotes the dissemination of coding practices and specific code knowledge throughout the organization. Meta uses Phabricator[2] as the cornerstone of its CI system. This platform facilitates contemporary code reviews, wherein developers submit code changes, that are referred to as *diffs* at Meta, and provide feedback on their peers' changes before they are either integrated into the codebase or rejected.

Phabricator and version control systems are used as part of the development process. Phabricator tracks code diffs, author/reviewer actions, and the current state of all diffs. Developers submit their code for review, creating a *patch* representing the initial version of the code. Reviewers can suggest improvements, leading to additional revisions until the diff is either approved and incorporated into the code base (the diff "lands"), or until the diff is abandoned. A diff is a collection of patches representing the initial version of a

[1] An A/B test is an industry jargon for (and we use it interchangeably with) a Randomized Controlled Trial (RCT) where diffs are assigned treatment condition (in this case workflow changes) or control condition (no changes). This chance assignment creates similar groups, minimizing bias and making it easier to prove if the treatment actually causes an effect, making RCTs the "gold standard" for medical evidence.

[2] http://phabricator.org

bug fix or enhancement, along with any revisions made during the diff's lifecycle.

2.2 DRS Model

Diff Risk Score (DRS)[3] is an AI-driven system developed at Meta designed to estimate the probability that a code modification will trigger a production incident, commonly referred to as a SEV. Built upon a fine-tuned Llama large language model (LLM), DRS analyzes code changes along with associated metadata to generate a risk score and identify potentially hazardous code segments. Currently, DRS supports numerous risk-sensitive features that enhance product quality, boost developer efficiency, and improve computational resource utilization. Importantly, DRS has enabled Meta to remove significant code freeze periods, allowing developers to deploy code in situations where they previously could not, with minimal adverse effects on user experience and business outcomes.

Modern state-of-the-art generative LLMs have proven effective in supporting various software engineering tasks, including code completion [1, 20, 44], test case generation [70], and code review [23, 45]. These models are built on a foundational model pretrained on vast internet-scale datasets (billions of tokens) using a straightforward generic objective, namely next token prediction. They are then either prompted directly or further fine-tuned on domain-specific datasets to improve performance on targeted tasks.

CodeLlama [67] is an example of such a model, developed on top of Meta GenAI's Llama2 [76] and fine-tuned specifically for code generation and discussion. CodeLlama offers enhanced capabilities for coding tasks, including generating code and natural language explanations from both code and natural language inputs (e.g., "Write me a function that outputs the Fibonacci sequence"). It also supports code completion and debugging across many widely used programming languages.

DRS utilizes a model derived from CodeLlama, called iDiffLlama [2]. Beyond pre-training on code and natural language, iDiffLlama is additionally pre-trained on internal diffs, i.e., code changes. This pre-training makes the model "change-aware," enabling it to comprehend not only static code but also the dynamics of code evolution through changes. Specifically, diffs represent a developer's intent to commit modifications to the monorepo. iDiffLlama learns this intent by modeling the relationships between a diff's title, summary, test plan, and the actual code changes.

We perform a supervised fine-tuning (SFT) phase on the pretrained LLM using labeled DRS data. To convert the classification task into a generative one, we annotate the input with special markers `[DRS][/DRS]`, append the label (0 or 1) at the end, and train the model to "generate" the label token. A detailed description of the model can be found at [2].

At inference time, we prompt the model to generate the label token. To obtain a risk score rather than just a label, we extract the probabilities of the label tokens "0" and "1" from the next token distribution. If the model is well-aligned, all other tokens in the vocabulary should have near-zero probability.

This approach allows the LLM to backpropagate and learn the subtleties of diff risk, effectively making it "risk-aligned," rather than relying on untuned embeddings that only capture general diff characteristics. This is the model we use to identify risky diffs in this paper.

[3] https://engineering.fb.com/2025/08/06/developer-tools/diff-risk-score-drs-ai-risk-aware-software-development-meta/

3 Developing the Risky Diff Reviewer (RDR) Workflow

We developed the Risky Diff Reviewer (RDR) workflow to focus reviewer attention on the small fraction of diffs that are flagged as high risk during review. The workflow builds on the Diff Risk Score (DRS), an internal machine learning signal that estimates the likelihood that a diff will trigger an incident after landing. When a diff's risk score exceeds the high-risk threshold (95th percentile), the code review UI surfaces a warning and highlights potentially risky snippets to help authors and reviewers quickly identify areas that merit closer inspection [2].

Figure 1 illustrates how RDR is triggered for a high-risk diff. When the warning appears, RDR automatically assigns an additional reviewer, called the *DRS Reviewer*, selected by the reviewer recommendation system. This reviewer is the top recommended candidate based on signals such as recent engagement with the relevant files and recent collaboration patterns [63]. In addition to surfacing the risk signal, RDR adds lightweight structure to the review process by making the recommended expert reviewer explicit and by recording key author and reviewer actions related to risk resolution.

Authors can clear the high-risk signal in two common ways. They can obtain approval from the DRS Reviewer (or another highly recommended reviewer), or they can add a mitigation plan that supports safe rollback, such as gating mechanisms like Gatekeepers, JustKnobs, or Quick Experiments.

Figure 2 shows the pre-land dialog that appears when an author attempts to land a high-risk diff without resolving the DRS signal. The dialog acts as a soft block: authors can proceed, but they provide a brief reason (for example, an urgent incident fix or confirmation of expert review). RDR also files a land issue when approvals from the DRS Reviewer or the top recommendations are missing, and it logs bypass actions for later analysis.

We also support reassignment when the default DRS Reviewer is not a good fit. Authors and reviewers can reassign the DRS Reviewer to someone with more relevant context, and the UI captures a structured reason such as *not available* or *does not have context*. We treat reassignment feedback, especially *does not have context*, as a signal for improving the recommendation system over time.

In summary, RDR is a workflow contribution that couples a UI-surfaced risk signal with targeted reviewer assignment, a lightweight pre-land prompt, and structured feedback loops. This combination provides a practical way to encourage deeper review on high-risk diffs while keeping the standard review experience unchanged for the majority of diffs. We evaluate this workflow in an RCT in Research Question 1.

4 Creating a New Recommender for RDR

The RDR workflow assigns an additional reviewer for high-risk diffs, so the quality of reviewer recommendation directly affects how RDR is experienced. We received feedback that the existing recommender was often not selecting the best reviewer for risky

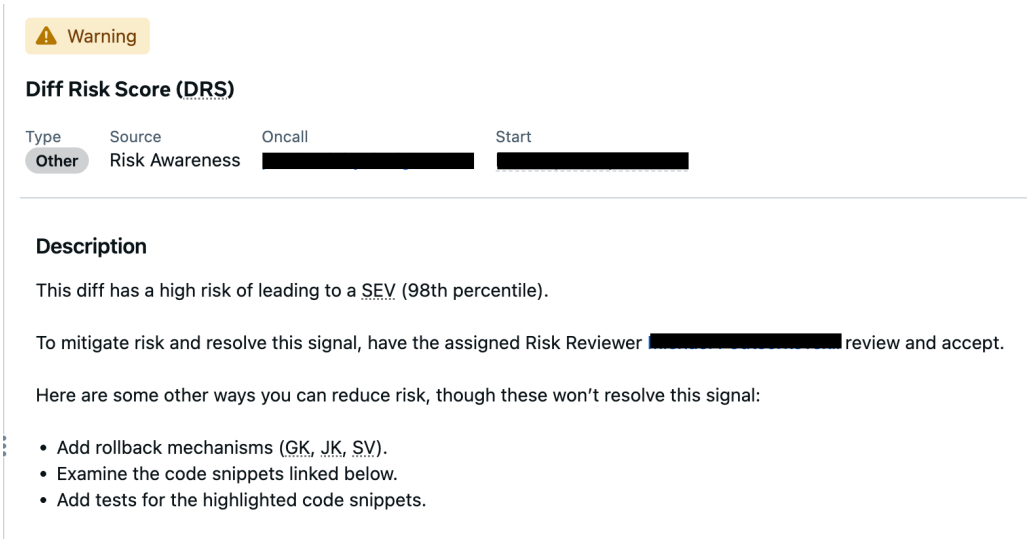

Figure 1: A warning signal appears on the diff, which indicates the risk level of the diff and the additional Risk Reviewer that is recommended to review the diff.

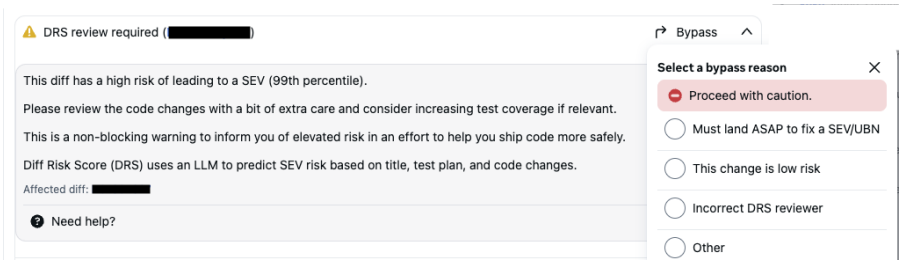

Figure 2: This pop-up dialogue appears on the diff when the author tries to land the diff while the current diff version is still high risk and while there is no approval from the Risk Reviewer. The dialogue can be bypassed by the author by indicating a reason for bypass.

diffs. This is consistent with the legacy model's original objective: it was tuned to reduce review latency by predicting who would take *any* action, using a limited feature set and a binary ground truth derived from historical reviewer actions [63]. For RDR, we instead focus on recommending reviewers with strong and *recent* context on the relevant code and collaboration patterns.

Following prior work, we used LambdaMART to train our model [63]. Since reviewer recommendation is a ranking problem, we select the learn-to-rank approach [12], LambdaMART. A pairwise comparison of each reviewer is made and this ordering is then used to determine the final order. In our case we use MART (Multiple Additive Regression Trees) as the function to determine the ordering of reviewers. LambdaMART is the boosted tree version of LambdaRank, which is based on RankNet. All have proven to be very successful algorithms for solving real world ranking problems. An approximate description of how it works would take more space than this paper and is presented in [12]. There is infrastructural support for LambdaMART models at scale using measures collecting data from internal databases on code review and coding.

4.1 Ground Truth for DRS Reviewers

The legacy model treated each candidate as a positive example if they took an action on the diff. This binary label is easy to construct, but it does not distinguish between shallow and substantive engagement, and it can over-emphasize availability and historical assignment patterns.

For RDR, we train on a continuous, weighted ground truth that scores each candidate based on multiple dimensions of reviewer fit in Table 2: file expertise (authorship and review experience on relevant files), author collaboration (attention and prior working relationship), and recent interaction history (comments and review actions on overlapping files), with recency decay to emphasize current context. We also include small adjustments that reward consistently engaging when tagged and penalize frequent resignations, so the label reflects meaningful participation rather than activity volume.

4.2 Model Features and Importance

To support this ground truth at review time, we expand beyond the existing activity-focused signals and add low-latency features that

Table 1: The existing recommender, RevRec, was trained on the first set of features. We added the bottom features to the new model that is trained for risky diff review, RecRDR. We only include features that can be calculated with low latency, we use the feature categories suggested by Al-Zubaidi *et al.* [3].

Category	Feature	Description
Familiarity with Author	Reviewer On Author Eyeball	The time the candidate spent in the diff review tool looking at the author's diffs in the past 90 days
Familiarity with Author	Assigned Reviewer On Author Eyeball	The time the candidate spent in the diff review tool looking at the author's diffs when they were explicitly assigned as a reviewer in the past 90 days
Review Participation Rate	assigned reviews	Number of diffs the candidate was assigned as reviewer. Reviewers are assigned to many diff either directly or as part of a team or review group.
Review Participation Rate	resigned diffs	Number of diffs the candidate resigned from. When a reviewer does not feel qualified to review a diff they can resign from that review.
Reviewing Experience	rejected diffs	Number of diffs the candidate rejected
Reviewing Experience	reviewer comments	Number of diffs the candidate commented or acted on as a reviewer
Reviewing Experience	total comments	Number of diffs the candidate commented on
Code Ownership	authored diffs	Number of diffs the candidate authored in the past which touched any of the files in the diff.
Reviewer Recency	recency strength	Exponential decay based on most recent interaction with interaction type weighting
Recent Comment Interaction Rate	author-reviewer comment interaction count	Total number of comments exchanged between the candidate and the author on previous diffs containing overlapping files in the last 180 days.
Recent Reviewer Actions Interaction Rate	author-reviewer review interaction count	Total number of explicit reviewer actions taken by the candidate for the author, and by the author for the candidate, on previous diffs containing overlapping files in the last 180 days.

Table 2: Ground truth for RecRDR model. Typically, code reviewer recommenders are only evaluated on how well they predict the person who made a comment or did an action during a code review. Our goal, is to focus the model on a diverse set of ground truth metrics with the weighting focusing on our goal of identifying good reviewers of risky diffs.

Measure	Description	Weight
Author collaboration	Quantifies overall attention the candidate has given to the author's diffs based on eyeball time.	35%
Core file experience	Measures how much the candidate has authored or reviewed diffs on the relevant files.	25%
Active engagement quality	Captures the candidate's engagement through comments on relevant diffs.	15%
Comment collaboration	Measures collaborative commenting activity between candidate and author.	10%
Review collaboration	Measures collaborative review/accept activity between candidate and author	10%
Recency factor	Measures how recent the collaboration or activity between candidate and author.	5%
Total	Sum of weights	100%

capture recency and depth of reviewer-author-file relationships. The full set of features is shown in Table 1. Key additions include *recency strength* (exponential decay over recent interactions) and explicit author-reviewer interaction counts over a recent window, alongside existing signals such as file experience and commenting behavior.

This change shifts the model toward substantive engagement signals. As shown in Table 5, RecRDR places substantially more weight on features that reflect active reviewing and collaboration (e.g., total comments and assigned-reviewer eyeball time), while the baseline RevRec relies more heavily on assignment-driven activity (e.g., assigned reviews and general eyeball time). This aligns recommendation behavior with the RDR goal of surfacing reviewers who actively understand and engage with the relevant code, rather than reviewers who are merely likely to act. In RQ2 we evaluate the new RecRDR on an offline backtest, and in RQ3 we contrast RevRec with RecRDR in an RCT.

5 Experimental Design

Most empirical studies in software engineering are observational, which makes causal inference difficult when treatment correlates with unobserved factors. We use Randomized Controlled Trials (RCTs), commonly called A/B tests in software engineering, to estimate causal effects by randomly assigning diffs to treatment and control [17, 29, 31].

We ran two RCTs to evaluate (1) the RDR workflow for high-risk diffs and (2) a new reviewer recommendation model within the RDR experience.

Experiment 1: RDR workflow. We ran a five-week A/B test on the top 5% of riskiest diffs, as identified by the Diff Risk Score (DRS)

Table 3: We conduct two experiments. We show the experimental condition, *e.g.,* the number of diff in each experiment. The table also shows the metrics we use to quantify the impact and success/failure of an experiment as goal, tracking, and guardrail metrics.

Experimental Conditions	**Description**	**Expt. 1**	**Expt. 2**
Experimental unit	The number of diffs in the experiment (50/50 randomized split)	15,794 diffs	5,372 diffs
Time period	Number of weeks the experiment ran	5 weeks	2 weeks
Risk threshold	The threshold at which a diff is considered by be high risk	5%	2.5%
Evaluation Metrics	**Description**	**Expt. 1**	**Expt. 2**
Abandon Rate	The rate at which diffs are abandoned	Goal	Guardrail
Rate of Final DRS Score Being Low Risk	The rate at which the landed version of a diff drops below the high risk threshold	Goal	Guardrail
Number of Versions	Number of published diff versions	Tracking	Tracking
Number of Long Comments	Number of comments by reviewers with ≥ 200 characters	Tracking	Tracking
Number of Comments	Number of comments by reviewers with ≥ 200 characters	Tracking	Tracking
Number of Commenters	Number of unique reviewer commenters	Tracking	Tracking
Number of Rejections	Number of diff rejections	Tracking	Tracking
Reviewer Tenure	Number of years since the reviewer was hired	Tracking	Tracking
Reviewer Eyeball Time	The active time spent by reviewers reviewing the diff	Tracking	Tracking
Diff Processing Time (DPT)	The time from when a diff is first published to when it lands	Guardrail	Guardrail
Recommended Reviewer *Commented*	The number of comments made by the recommended reviewer	NA	Goal
Recommended Reviewer *Accepted*	The number of diffs accepted by the recommended reviewer	NA	Goal

model. Eligible diffs were randomly assigned by Phabricator diff ID to control or treatment (50/50). Control diffs followed the standard review workflow. Treatment diffs used RDR: a DRS reviewer was automatically assigned, and a bypassable pre-land dialog prompted additional scrutiny before landing. The experiment included 7,981 control diffs and 7,813 treatment diffs.

Experiment 2: recommender model within RDR. We later ran a two-week A/B test on the top 2.5% of riskiest diffs. Eligible diffs were randomly assigned by Phabricator diff ID to RDR with the baseline recommender (RecV1) or RDR with the DRS-tuned model (RecRDR), with no other changes to the RDR experience. The experiment included 2,756 control diffs and 2,616 treatment diffs.

We tracked the metrics in Table 3 for each experiment. Goal metrics focus on review quality proxies (reviewer actions and reviewer characteristics), and we also report tracking metrics to capture related review activity. As a guardrail, we track Diff Processing Time (DPT), measured as the time from when a diff is published to when it is closed.

6 Results

6.1 RQ 1. Risky Diff Reviewer (RDR) Experiment

Expt. 1: Can we drive more thorough review by more qualified reviewers on risky diffs?

Our DRS model is capable of quantifying the risk of a diff [2]. Using the A/B test methodology described in Section 5, we rollout the RDR flow using existing RevRec to recommend an extra reviewer on the top 5% of risky diffs and contrast it with the existing workflow for diff review. In Table 4, we discuss the impact on each of our goal, tracking, and guardrail metrics.

For the goal metric, *Abandonment Rate*, we see a statistically significant increase of 13% more abandonments in the RDR condition, which implies that authors are being more cautions with their changes. Interestingly, we do not see a statistically significant increase ($p = .44$) in the *Number of Rejections* because rather than forcing a reject on the authors, the reviewers comments will allow the author to decide to abandon the change.

For the goal metric, *Rate of Final DRS Score Being Low Risk.*, when a diff is flagged as risky, authors examine their code and often reduce the risk by reworking the diff so that it is below the threshold, which unblocks the diff and avoids having to get an additional risk reviewer. In the RDR condition, 9.56% more diffs

Table 4: RQ1. Expt 1 Results. We compare the current review workflow with the additional risk reviewer RDR workflow that use the RecRDR. Our goal metrics indicate that the additional reviewer reduced the final risk score and lead to more comments from reviewers. These are promising proxy measures of improvement in review quality.

Metric Name	**Metric Type**	**% Difference**	**p-value**
Abandon Rate	Goal	13.09%	0.01
Rate of Final DRS Score Being Low Risk	Goal	9.56%	0.00
Number of Versions	Tracking	4.89%	0.00
Number of Long Comments	Tracking	9%	0.00
Number of Comments	Tracking	6.18%	0.00
Number of Commenters	Tracking	5.63%	0.00
Number of Rejections	Tracking	—	0.44
Reviewer Tenure	Tracking	2.21%	0.00
Reviewer Eyeball Time	Tracking	0.97%	0.00
Diff Processing Time (DPT)	Guardrail	0.45%	0.02

have a final risk score below the threshold. This drop is important because authors and reviewers take steps to reduce the risk, which is then confirmed by the change in score showing leading to fewer high risk diffs being rolled into production.

Our **Tracking Metrics** add additional insights related to our goal metrics. The increase is expected in each as the RDR condition results in a blocked diff and the assignment of an additional reviewer. The *Number of Commenters, Comments, and Long Comments* all increase by 5.6%, 6.1%, and 9%, respectively. These metrics show an increased discussion of high risk diffs flagged in the RDR condition. We also see more experienced reviewers with higher 2.2% *Reviewer Tenure* and reviewers spending more time on the diff *Active Review Time* increase of 0.9%. These comments and additional expertise lead to an increase in the *Number of Versions* of 4.8% as the authors refine the diff more frequently with RDR.

With the addition of a reviewer and indication of risk, we did not want to see a substantial increase in *Diff Processing Time (DPT)*, our **Guardrail metric**. We see a modest 0.45% increase in DPT. This indicates that reviews do take longer, but these are our risky diffs and there increase is minor.

> Expt. 1: We are able to drive significant improvements in proxy measures of review quality, *e.g.,* more comments by reviewers with greater expertise and a reduction in final DRS score, with less than a 0.5% increase in the review diff processing time.

6.2 RQ 2. A recommender model for risky diffs

Can we more accurately recommend the reviewers for risky diffs?

In the previous experiment, we saw that flagging the riskiness of a diff and recommending an extra reviewer improved the thoroughness of the review and reduced the risk score for the diffs. However, engineers complained that our recommendations were often not the best reviewer. Reviewer recommenders are typically trained on a ground truth that simply know the historical reviewers that took an action on a diff, *e.g.,* a comment or accept. In Section 4, we developed a new ground truth that is weighted towards the characteristics of reviewers that engineers for those reviewing risky diffs.

In Section 4, we describe the model and training. In Table 5 that we see the change in feature importance for the original model, RevRec vs the new model that is trained on features relevate to predicting the best reviewer for a risky diff. We see that drastically different feature importance in Table 5. In the RevRec model, the most influential features were reviewer-on-author eyeball time (33.89%) and the number of assigned reviews (31.28%), indicating a strong reliance on general reviewer activity and assignment patterns. In contrast, the RecRDR model, which was specifically tuned for high-risk diffs, shifted its emphasis: total comments became the most important feature (28.04%), followed by reviewer-on-author eyeball time (24.26%).

This shift in feature importance is a direct consequence of the change in ground truth, from a binary label to a weighted, multi-dimensional score, which allows the model to better capture the nuanced qualities of effective code review. As a result, features that reflect substantive reviewer engagement, such as commenting, are now more strongly prioritized, aligning the model's behavior with the goal of improving review quality and risk mitigation for the riskiest code changes.

As a guardrail/safety test, we need to make sure that the new recommender is still recommending reviewers that took action on a diff. We see in Table 6 that the RecRDR actually improves upon the original RevRec in terms of predicting who will take an action. Given that the new model, RecRDR did not regression on Top1 or Top3 accuracy, we are now ready to roll the new model into an A/B test.

Table 5: RQ2. Feature importance. We order the features by their importance to RecRDR and contrast it with the prior recommender RevRec. We note that different ground truths are used in training and this has a major impact on feature importance. For example, the top feature for RevRec was the amount of time the reviewer spent reviewing prior diffs from the author. In contrast, total comments is the most important feature for RecRDR as the ground truth is related to overall reviewer prominence.

Recommender	RecRDR (new)	RevRec (old)
Feature	**Importance**	**Importance**
total comments	38.04%	1.57%
assigned reviewer on author eyeball time	24.26%	6.69%
assigned reviews	12.93%	31.28%
reviewer on author eyeball time	11.55%	33.89%
authored diffs	6.08%	4.91%
reviewer comments	2.17%	10.10%
recency strength	1.82%	NA
author-reviewer comment interaction count	1.09%	NA
rejected diffs	0.66%	11.56%
author-reviewer review interaction count	0.62%	NA
resigned diffs	0.00%	0.00%

Table 6: RQ2. Improvement in Accuracy for RecRDR over RevRec. To ensure that we did not regress on the accuracy of the recommendations, we backtested the recommenders on an unseen set of code reviews based and used the person who actual did the review as the ground truth.

Recommender	Top-1	Top-3
RevRec	41.28%	51.56%
RecRDR	44.42%	54.76%
Improvement	3.14 points	3.20 points

Table 7: RQ 3. Expt. 2 compared RevRec with RecRDR. We see that RecRDR improved our goal metrics with the recommended reviewer making 9.9% more comments and accepting 7.7% more diffs than the RevRec recommended reviewer. This improvement shows that RecRDR provides better recommendations for risky diff review in production.

Metric Name	Metric Type	% Difference	p-value
Abandon Rate	Guardrail	—	0.10
Rate of Final DRS Score Being Low Risk	Guardrail	—	0.59
Number of Versions	Tracking	—	0.56
Number of Long Comments	Tracking	—	0.88
Number of Comments	Tracking	—	0.18
Number of Commenters	Tracking	6.17%	0.01
Number of Rejections	Tracking	—	0.32
Reviewer Tenure	Tracking	—	0.84
Reviewer Eyeball Time	Tracking	10.79%	0.04
Diff Processing Time (DPT)	Guardrail	—	0.15
Recommended Reviewer Commented	Goal	9.94%	0.01
Recommended Reviewer Accepted	Goal	7.77%	0.00

New model: The new ground truth drastically changed the feature importance focusing RecRDR on the features that are important for risky diff review. We also see that the new RecRDR model did not regress on the prior accuracy measure of prediction which reviewers were able to comment and act on diffs. We concluded that RecRDR was ready for an A/B test.

6.3 RQ 3. RevRec vs RecRDR Experiment

Expt. 2: How well does the new recommender work in practice?

Our first experiment showed that adding reviewers to risky diffs lead to more thorough reviews and a reduction in risk scores. However, the existing recommender, RevRec, often provided recommended reviewers that engineers indicated were not the best reviewer. RecRDR was trained to find the best reviewer for a risky diffs. We roll out an experiment that compares RevRec vs RecRDR in a double blind trial A/B test.

Table 7 displays the results from the experiment 2. Our goal was to make better reviewer recommendations for risky diffs that were quantified as the number of comments the recommended reviewer made and the number of diffs the recommended reviewer accepted. We see a statistically significant improvement, 9.9% and 7.7%, respectively for both goal metrics.

For the tracking metrics, only the number of commenters increased by 6.17% in a statistically significant manner. This is likely a byproduct of the extra reviewer being somehow could more likely do the review and is tied to the goal metrics.

We did not want to see regressions in our guardrail metrics. We see that the abandonment rate and the rate in diffs with lower risk scores did not change at statistically significant levels. We also did not see an increase in the amount of time to process and review a diff (DPT).

On the basis of these results, we replaced RevRec with RecRDR for the top 5% of risky diffs.

Expt. 2: We substantially improved the amount of time, number of comments, and number of diffs accepted by the reviewer recommended by the RecRDR. We did not see any regressions in our guardrails and continue to see a similar drop in the final DRS score from adding reviewers. RecRDR is now used on all risky diffs.

7 Literature and Discussion

We position our findings in the research literature and discuss how we advance our understanding of (1) risk-aware code review workflows for high-risk changes, and (2) reviewer recommendation models that are tuned to current expertise and evaluated in RCTs.

7.1 Code Review Practice and Risk

Fagan's seminal work on software inspections [21] demonstrated that structured reviews can detect defects early and reduce downstream effort. Subsequent work refined inspection processes and quantified their benefits and sources of variation [22, 55, 56]. However, the synchronous, meeting-heavy nature of traditional inspections has limited their adoption in fast-moving engineering environments [6].

Modern code review has largely replaced formal inspections with lightweight, tool-supported, asynchronous review workflows [6, 60–62]. Studies of open source and industrial projects show that modern review supports defect finding, knowledge sharing, and process conformance [10, 11, 37, 43, 46, 53, 69]. At the same time, review quality and participation are influenced by human and organizational factors such as workload, seniority, and motivation [9, 26, 68].

In parallel, defect prediction and risk modeling have been widely studied [28, 32, 32, 35, 50, 72, 75, 78, 82]. Traditional models rely on static and process metrics to identify risky components or commits prior to release, while more recent work leverages deep learning and behavioral traces for just-in-time defect prediction [32, 50]. Recent industrial work integrates risk models into deployment pipelines, using predictive signals to manage code freezes and rollout decisions [2, 49].

Our Diff Risk Score (DRS) system [2, 49] fits into this line of work as a just-in-time risk signal for individual diffs, powered by an LLM trained on code changes. Unlike most prior defect prediction studies, which are evaluated offline using metrics such as AUC or F1, we study how exposing a risk signal directly in the review UI and

coupling it with workflow interventions affects review behavior and measured risk on high-risk diffs in production.

7.2 Defect Detection and Risk Mitigation During Review

Code modifications introduced during development and maintenance can lead to defects that impact quality and reputation [8, 48, 54, 77]. Modern code review plays a central role in detecting such defects before release [37, 39, 46, 60]. Empirical studies show that review measures (e.g., number of comments, reviewer experience, participation) correlate with post-release defects [37, 39, 46], but causality is difficult to establish.

Kazemi et al. propose the Changeset Safety Ratio (CSR) to quantify the risk of inducing defects after code integration and use it to study the effect of reviewer recommendation on risk [33]. They simulate reviewer replacement by substituting one reviewer with a recommended expert and show that expertise-based recommenders can reduce risk but do not explicitly address knowledge distribution or turnover. Other work measures outcomes for reviewer recommenders beyond accuracy, such as mitigating knowledge loss or balancing workload [5, 27, 47, 59], often using historical simulations instead of RCTs.

In contrast to replacement-based strategies, we focus on live interventions that shape review behavior on the riskiest diffs. Our first research question (RQ1) asks whether a dedicated, risk-aware workflow—automatically assigning a top reviewer and adding a soft-blocking pre-land dialog—drives more thorough and expert review on high-risk diffs. We measure changes in review depth (comments, long comments, versions, unique commenters), reviewer expertise, and the final DRS risk score, while guarding developer velocity via diff processing time. Our results show that for the top 5% riskiest diffs, the Risky Diff Reviewer (RDR) workflow leads to more cautious author behavior (e.g., higher abandonment), deeper and more collaborative reviews, and about a 10% relative increase in the proportion of diffs that land below the high-risk threshold, with less than a 0.5% increase in processing time.

7.3 Code Reviewer Recommendation

Identifying suitable reviewers for a code change is a central challenge in modern code review [6, 7, 21, 26, 85]. Inappropriate reviewer selection can slow down reviews [74] or reduce review quality [6, 11, 37]. Reviewer recommendation systems therefore aim to automatically assign review requests to developers who are likely to provide expert and timely feedback [7, 13, 16].

Prior work has explored a wide range of techniques, including heuristic-based scoring over file and author history [5, 27, 36, 47, 58, 74, 85], search-based and multi-objective optimization [3, 15, 51, 59], collaborative filtering and matrix factorization [16, 80], machine learning and learning-to-rank [4, 30, 52, 73], text and topic mining [40–42, 79], and graph- and hypergraph-based models over multiplex relationships [42, 57, 65, 81, 83, 84, 86]. Industrial deployments at Microsoft, Tencent, and other organizations show that reviewer recommenders can reduce latency and manual effort at scale [5, 14, 86].

Most of this work evaluates recommenders offline by comparing the recommended list to the historical set of reviewers who actually reviewed a change [13, 34]. This evaluation setup implicitly assumes that historical reviewers are ideal and that reproducing their choices is the primary goal. However, empirical studies question this assumption: Kovalenko et al. show that developers often already know the top recommended reviewers and that Top-K accuracy and MRR provide limited additional value [38]; Gauthier et al. and Kazemi et al. find that recommended but unused reviewers would often have been appropriate choices and that recommendations can become stale over time [24, 34]. Doğan et al. explicitly investigate the validity of ground truth for reviewer recommendation and highlight systematic biases in labeling based solely on historical participation [18].

Recent work therefore broadens the goals of reviewer recommendation to include balancing workload, mitigating turnover-induced knowledge loss, and improving knowledge distribution [27, 47, 48, 59, 64]. Mirsaeedi and Rigby [47] evaluate recommenders in terms of expertise, core team workload, and files-at-risk-to-turnover, while Hajari et al. use the Gini coefficient to measure workload distribution [19, 27]. Industrial systems such as WhoDo [5] also track operational metrics like average number of open reviews and completion time.

Our work contributes to this literature in three ways. First, for RQ2 we redefine the ground truth used to train a reviewer recommendation model for high-risk diffs. Instead of a binary label indicating whether a candidate reviewed a diff, we construct a continuous, weighted score over file expertise, author collaboration, interaction quality, and recency, informed by concerns about stale recommendations and biased labels [18, 24, 34]. Second, we design a low-latency feature set that emphasizes recent, meaningful engagement (e.g., total comments, reviewer-on-author eyeball time, recent interaction counts) and show in offline backtests that the resulting RecRDR model improves Top-1 and Top-3 action rates over our legacy model, while shifting feature importance toward substantive engagement.

Third, and most importantly for RQ3, we evaluate reviewer recommendation for risky diffs in a production A/B test rather than only on historical data. We compare the legacy recommender to RecRDR within the same RDR workflow, using goal metrics focused on recommended reviewer behavior (comments and acceptances), tracking metrics on broader review activity, and guardrails on diff processing time and abandonment. RecRDR increases comments and acceptances from recommended reviewers by roughly 8–10% and 7–8%, respectively, without harming developer velocity. This complements prior industrial case studies [5, 14, 86] by showing that a risk-aware, recency-tuned recommender can measurably improve review engagement on the riskiest diffs, when coupled with a workflow that surfaces risk signals directly in the review UI.

8 Threats to Validity

8.1 Generalizability

We conducted the evaluation within a large engineering ecosystem, which includes Phabricator-based code review, a monorepo, internal telemetry, and the Diff Risk Score (DRS) signal that triggers the Risky Diff Reviewer workflow. These conditions may differ from

other organizations in release cadence, reviewer culture, code ownership, and the availability of low-latency signals. We rely on features such as eyeball time, recent author-reviewer interactions, and assignment heuristics; environments without similar instrumentation or review practices may not transfer these results directly.

We targeted only the top 5% (or 2.5% in one experiment) of diffs by DRS, which constrains our findings to the riskiest tail of changes. Effects may differ for medium-risk or routine diffs. We excluded some groups and non-config diffs at times, which may bias applicability to domains with different policies or incident profiles. Short evaluation windows (five weeks and two weeks) limit our inference about seasonal patterns, staffing cycles, and long-term reviewer learning effects.

8.2 Construct Validity

We use several proxy outcome measures for review quality and risk reduction. Counts of comments, long comments (length threshold), versions, unique commenters, reviewer tenure, and reviewer eyeball time indicate engagement, but they do not directly measure defect prevention or SEV avoidance. We observe a drop in the final DRS score, but this is a model output rather than an observed ground-truth risk outcome. Improvements in these proxies may not fully translate to fewer production incidents.

We assess reviewer recommendation accuracy via Top-1 and Top-3 action rates. We define action broadly (accept, comment, endorse, reject), and this definition may conflate high-quality review with minimal engagement. We build a continuous, weighted ground truth for RecRDR from internal signals (file experience, collaboration, recent interactions). These signals may encode historical assignment biases or code ownership norms, which can skew the label toward entrenched reviewers rather than emerging experts. Differences in tool usage and parallel responsibilities can affect eyeball time and interaction counts; these factors introduce measurement noise.

We use Diff Processing Time (DPT) as a guardrail for velocity, but it mixes review latency with author iteration time, test execution, and landing windows. UX changes (soft-block dialogs, reassignment options) may alter behaviors independent of review quality, which complicates attribution. Authors self-report bypass reasons, and these reports may be incomplete or strategic, which reduces the precision of behavioral interpretations.

8.3 Internal Validity

We randomize by diff ID to balance confounders, but cluster effects may remain. Reviewers may participate across control and test diffs, which creates spillover. Authors or teams with concentrated risk profiles can induce correlation within clusters.

Flagging risk can change behavior. Reviewers and authors may respond to the signal because they know the diff is flagged, rather than due to the recommended reviewer or the model's quality. Reviewer availability and workload fluctuate, which affects who acts on a diff independent of the recommendation. We capture reassignment reasons, but the feature also creates an intervention channel that can modify reviewer composition independent of the model.

9 Conclusion and Contributions

We study whether risk-aware interventions can improve code review outcomes for the riskiest code changes in a large, fast-moving engineering environment while preserving developer velocity. Our approach couples three components: (1) a UI-surfaced risk signal via the Diff Risk Score (DRS), (2) a lightweight workflow that focuses additional expert attention on high-risk diffs, and (3) a reviewer recommender tuned for risky diffs that prioritizes recent and relevant expertise.

We introduce the Risky Diff Reviewer (RDR) workflow, which automatically assigns an additional recommended reviewer when a diff is flagged as high risk and presents a bypassable pre-land dialog when authors attempt to land without resolving the risk signal. This workflow keeps the standard review experience unchanged for the majority of diffs while adding structure and visibility to review and mitigation behavior on the high-risk tail.

We evaluate RDR and our recommender changes using offline backtests and two A/B tests instrumented with goal, tracking, and guardrail metrics. In the workflow experiment on the top 5% of diffs by DRS (over 15k diffs), RDR increases collaborative review activity and iteration, and it increases the fraction of diffs whose final version drops below the high-risk threshold (+9.56%), with a modest change in Diff Processing Time (+0.45%). We also redesign reviewer recommendation for risky diffs by moving beyond a binary "did they review" label to a continuous, weighted ground truth that emphasizes file experience, author collaboration, interaction quality, and recency, using only low-latency features available at recommendation time. In offline evaluation, the resulting model (RecRDR) improves Top-1 and Top-3 action rates over the baseline. In an A/B test on over 5k diffs we compare the new and old recommender and find that RecRDR increases engagement from the recommended reviewer (+9.94% comments and +7.77% acceptances) without statistically significant regressions in guardrail metrics such as diff processing time.

We make three contributions. First, we present a practical, risk-aware code review workflow that couples a clear, UI-surfaced risk signal with targeted reviewer assignment and a lightweight pre-land prompt, and we provide experimental evidence that it improves engagement and reduces the fraction of diffs that remain above the high-risk threshold at landing. Second, we introduce a reviewer recommender tailored to risky diffs, including a continuous, weighted training signal and a low-latency feature set designed to emphasize recent and meaningful reviewer expertise and collaboration. Third, we demonstrate, through RCTs at scale, that these workflow and recommendation interventions improve reviewer engagement on high-risk diffs while maintaining velocity-focused guardrails.

In future work, we plan to connect these proxy improvements to longer-term outcomes beyond review activity and model scores, to explore additional risk bands, and to incorporate workload and coverage objectives into reviewer assignment and recommendation.

Acknowledgments

We used AI to perform editorial enhancements.

References

[1] Github Accessed 2026. *Github Copilot.* Github. https://github.com/features/copilot

[2] Rui Abreu, Vijayaraghavan Murali, Peter C Rigby, Chandra Maddila, Weiyan Sun, Jun Ge, Kaavya Chinniah, Audris Mockus, Megh Mehta, and Nachiappan Nagappan. 2025. Moving Faster and Reducing Risk: Using LLMs in Release Deployment. In *2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 448–457.

[3] Wisam Haitham Abbood Al-Zubaidi, Patanamon Thongtanunam, Hoa Khanh Dam, Chakkrit Tantithamthavorn, and Aditya Ghose. 2020. Workload-Aware Reviewer Recommendation Using a Multi-Objective Search-Based Approach. In *Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering* (Virtual, USA) *(PROMISE 2020)*. Association for Computing Machinery, New York, NY, USA, 21–30. doi:10.1145/3416508.3417115

[4] Ishan Aryendu, Ying Wang, Farah Elkourdi, and Eman Abdullah Alomar. 2022. Intelligent code review assignment for large scale open source software stacks. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–6.

[5] Sumit Asthana, Rahul Kumar, Ranjita Bhagwan, Christian Bird, Chetan Bansal, Chandra Maddila, Sonu Mehta, and Balasubramanyan Ashok. 2019. Whodo: Automating reviewer suggestions at scale. In *Proceedings of the 2019 27th ACM joint meeting on European Software Engineering Conference and symposium on the foundations of Software Engineering*. 937–945.

[6] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 international conference on software engineering*. IEEE Press, 712–721.

[7] Vipin Balachandran. 2013. Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 931–940.

[8] Rajiv D Banker, Srikant M Datar, Chris F Kemerer, and Dani Zweig. 1993. Software complexity and maintenance costs. *Commun. ACM* 36, 11 (1993), 81–95.

[9] Amiangshu Bosu and Jeffrey C Carver. 2013. Impact of peer code review on peer impression formation: A survey. In *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. IEEE, 133–142.

[10] Amiangshu Bosu, Jeffrey C Carver, Christian Bird, Jonathan Orbeck, and Christopher Chockley. 2016. Process aspects and social dynamics of contemporary code review: Insights from open source development and industrial practice at microsoft. *IEEE Transactions on Software Engineering* 43, 1 (2016), 56–75.

[11] Amiangshu Bosu, Michaela Greiler, and Christian Bird. 2015. Characteristics of useful code reviews: An empirical study at microsoft. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 146–156.

[12] Christopher JC Burges. 2010. From ranknet to lambdarank to lambdamart: An overview. *Learning* 11, 23-581 (2010), 81.

[13] H Alperen Çetin, Emre Doğan, and Eray Tüzün. 2021. A review of code reviewer recommendation studies: Challenges and future directions. *Science of Computer Programming* (2021), 102652.

[14] Qiuyuan Chen, Dezhen Kong, Lingfeng Bao, Chenxing Sun, Xin Xia, and Shanping Li. 2022. Code Reviewer Recommendation in Tencent: Practice, Challenge, and Direction. In *2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 115–124. doi:10.1145/3510457.3513035

[15] Moataz Chouchen, Ali Ouni, Mohamed Wiem Mkaouer, Raula Gaikovina Kula, and Katsuro Inoue. 2021. WhoReview: A multi-objective search-based approach for code reviewers recommendation in modern code review. *Applied Soft Computing* 100 (2021), 106908.

[16] Aleksandr Chueshev, Julia Lawall, Reda Bendraou, and Tewfik Ziadi. 2020. Expanding the number of reviewers in open-source projects by recommending appropriate developers. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 499–510.

[17] Thomas D Cook, Donald Thomas Campbell, and William Shadish. 2002. *Experimental and quasi-experimental designs for generalized causal inference*. Vol. 1195. Houghton Mifflin Boston, MA.

[18] Emre Doğan, Eray Tüzün, K Ayberk Tecimer, and H Altay Güvenir. 2019. Investigating the validity of ground truth in code reviewer recommendation studies. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 1–6.

[19] Robert Dorfman. 1979. A formula for the Gini coefficient. *The review of economics and statistics* (1979), 146–149.

[20] Omer Dunay, Daniel Cheng, Adam Tait, Parth Thakkar, Peter C Rigby, Andy Chiu, Imad Ahmad, Arun Ganesan, Chandra Maddila, Vijayaraghavan Murali, Ali Tayyebi, and Nachiappan Nagappan. 2024. Multi-line AI-assisted Code Authoring.

[21] M. E. Fagan. 1976. Design and Code Inspections to Reduce Errors in Program Development. *IBM Systems Journal* 15, 3 (1976), 182–211.

[22] Michael E Fagan. 2001. Advances in software inspections. In *Pioneers and Their Contributions to Software Engineering*. Springer, 335–360.

[23] Alexander Frömmgen, Jacob Austin, Peter Choy, Nimesh Ghelani, Lera Kharatyan, Gabriela Surita, Elena Khrapko, Pascal Lamblin, Pierre-Antoine Manzagol, Marcus Revaj, et al. 2024. Resolving code review comments with machine learning. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*. 204–215.

[24] Ian X Gauthier, Maxime Lamothe, Gunter Mussbacher, and Shane McIntosh. 2021. Is historical data an appropriate benchmark for reviewer recommendation systems?: A case study of the gerrit community. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 30–41.

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

[26] Michaela Greiler, Christian Bird, Margaret-Anne Storey, Laura MacLeod, and Jacek Czerwonka. 2016. Code Reviewing in the Trenches: Understanding Challenges, Best Practices and Tool Needs.

[27] Fahimeh Hajari, Samaneh Malmir, Ehsan Mirsaeedi, and Peter C Rigby. 2024. Factoring Expertise, Workload, and Turnover into Code Review Recommendation. *IEEE Transactions on Software Engineering* (2024).

[28] Ahmed E Hassan. 2009. Predicting faults using the complexity of code changes. In *2009 IEEE 31st international conference on software engineering*. IEEE, 78–88.

[29] Alehandro R Jadad and Murray W Enkin. 2007. *Randomized controlled trials: questions, answers and musings*. John Wiley & Sons.

[30] Gaeul Jeong, Sunghun Kim, Thomas Zimmermann, and Kwangkeun Yi. 2009. Improving code review by predicting reviewers and acceptance of patches. *Research on software analysis for error-free computing center Tech-Memo (ROSAEC MEMO 2009-006)* (2009), 1–18.

[31] Natalia Juristo and Ana M Moreno. 2013. *Basics of software engineering experimentation*. Springer Science & Business Media.

[32] Yasutaka Kamei, Takafumi Fukushima, Shane McIntosh, Kazuhiro Yamashita, Naoyasu Ubayashi, and Ahmed E Hassan. 2016. Studying just-in-time defect prediction using cross-project models. *Empirical Software Engineering* 21, 5 (2016), 2072–2106.

[33] Farshad Kazemi, Maxime Lamothe, and Shane McIntosh. 2022. Exploring the Notion of Risk in Code Reviewer Recommendation. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 139–150.

[34] Farshad Kazemi, Maxime Lamothe, and Shane McIntosh. 2024. Characterizing the Prevalence Distribution and Duration of Stale Reviewer Recommendations. *IEEE Transactions on Software Engineering* (2024).

[35] Sunghun Kim, E James Whitehead, and Yi Zhang. 2008. Classifying software changes: Clean or buggy? *IEEE Transactions on software engineering* 34, 2 (2008), 181–196.

[36] Dezhen Kong, Qiuyuan Chen, Lingfeng Bao, Chenxing Sun, Xin Xia, and Shanping Li. 2022. Recommending code reviewers for proprietary software projects: A large scale study. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 630–640.

[37] Oleksii Kononenko, Olga Baysal, Latifa Guerrouj, Yaxin Cao, and Michael W Godfrey. 2015. Investigating code review quality: Do people and participation matter?. In *2015 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 111–120.

[38] Vladimir Kovalenko, Nava Tintarev, Evgeny Pasynkov, Christian Bird, and Alberto Bacchelli. 2018. Does reviewer recommendation help developers? *IEEE Transactions on Software Engineering* (2018).

[39] Andrey Krutauz, Tapajit Dey, Peter C Rigby, and Audris Mockus. 2020. Do code review measures explain the incidence of post-release defects? Case study replications and bayesian networks. *Empirical Software Engineering* 25, 5 (2020), 3323–3356.

[40] Ruiyin Li, Peng Liang, and Paris Avgeriou. 2023. Code reviewer recommendation for architecture violations: An exploratory study. In *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*. 42–51.

[41] Zhifang Liao, Zexuan Wu, Jinsong Wu, Yan Zhang, Junyi Liu, and Jun Long. 2019. TIRR: A code reviewer recommendation algorithm with topic model and reviewer influence. In *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 1–6.

[42] Jiahui Liu, Ansheng Deng, Qiuju Xie, and Guanli Yue. 2023. A code reviewer recommendation approach based on attentive neighbor embedding propagation. *Electronics* 12, 9 (2023), 2113.

[43] L. MacLeod, M. Greiler, M. Storey, C. Bird, and J. Czerwonka. 2018. Code Reviewing in the Trenches: Challenges and Best Practices. *IEEE Software* 35, 4 (2018), 34–42. doi:10.1109/MS.2017.265100500

[44] Chandra Maddila, Negar Ghorbani, Kosay Jabre, Vijayaraghavan Murali, Edwin Kim, Parth Thakkar, Nikolay Pavlovich Laptev, Olivia Harman, Diana Hsu, Rui Abreu, et al. 2025. AI-Assisted SQL Authoring at Industry Scale. In *2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 148–158.

[45] Chandra Maddila, Negar Ghorbani, James Saindon, Parth Thakkar, Vijayaraghavan Murali, Rui Abreu, Jingyue Shen, Brian Zhou, Nachiappan Nagappan, and Peter C Rigby. 2025. AI-Assisted Fixes to Code Review Comments at Scale. *arXiv preprint arXiv:2507.13499* (2025).

[46] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan. 2016. An Empirical Study of the Impact of Modern Code Review Practices on Software Quality. *Empirical Software Engineering* 21, 5 (2016), 2146–2189.

[47] Ehsan Mirsaeedi and Peter C. Rigby. 2020. Mitigating Turnover with Code Review Recommendation: Balancing Expertise, Workload, and Knowledge Distribution. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 1183–1195. doi:10.1145/3377811.3380335

[48] Audris Mockus. 2010. Organizational volatility and its effects on software defects. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*. ACM, 117–126.

[49] Audris Mockus, Rui Abreu, Peter C Rigby, David Amsallem, Parveen Bansal, Kaavya Chinniah, Brian Ellis, Peng Fan, Jun Ge, Bingjie He, et al. 2025. Leveraging Risk Models to Improve Productivity for Effective Code Un-Freeze at Scale. *ACM Transactions on Software Engineering and Methodology* (2025).

[50] Issei Morita, Yutaro Kashiwa, Masanari Kondo, Jeongju Sohn, Shane McIntosh, Yasutaka Kamei, and Naoyasu Ubayashi. 2024. TraceJIT: Evaluating the Impact of Behavioral Code Change on Just-In-Time Defect Prediction. In *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 580–591.

[51] Ali Ouni, Raula Gaikovina Kula, and Katsuro Inoue. 2016. Search-based peer reviewers recommendation in modern code review. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 367–377.

[52] Prahar Pandya and Saurabh Tiwari. 2022. CORMS: a github and gerrit based hybrid code reviewer recommendation approach for modern code review. In *Proceedings of the 30th ACM joint European software engineering conference and symposium on the foundations of software engineering*. 546–557.

[53] Luca Pascarella, Davide Spadini, Fabio Palomba, Magiel Bruntink, and Alberto Bacchelli. 2018. Information needs in contemporary code review. *Proceedings of the ACM on human-computer interaction* 2, CSCW (2018), 1–27.

[54] Kishan Patel and Sandeep Gupta. 2024. Prediction of Software Defects for Quality Assurance and Improvement Based on Machine Learning Methods. In *2024 2nd International Conference on Advances in Computation, Communication and Information Technology (ICAICCIT)*, Vol. 1. IEEE, 400–406.

[55] Dewayne E Perry, Adam Porter, Michael W Wade, Lawrence G Votta, and James Perpich. 2002. Reducing inspection interval in large-scale software development. *IEEE Transactions on Software Engineering* 28, 7 (2002), 695–705.

[56] Adam Porter, Harvey Siy, Audris Mockus, and Lawrence Votta. 1998. Understanding the sources of variation in software inspections. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 7, 1 (1998), 41–79.

[57] Yu Qiao, Jian Wang, Can Cheng, Wei Tang, Peng Liang, Yuqi Zhao, and Bing Li. 2024. Code Reviewer Recommendation Based on a Hypergraph with Multiplex Relationships. In *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 417–428.

[58] Mohammad Masudur Rahman, Chanchal K. Roy, and Jason A. Collins. 2016. CORRECT: Code Reviewer Recommendation in GitHub Based on Cross-Project and Technology Experience. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. 222–231.

[59] Soumaya Rebai, Abderrahmen Amich, Somayeh Molaei, Marouane Kessentini, and Rick Kazman. 2020. Multi-objective code reviewer recommendations: balancing expertise, availability and collaborations. *Automated Software Engineering* 27 (2020), 301–328.

[60] Peter Rigby, Brendan Cleary, Frederic Painchaud, Margaret-Anne Storey, and Daniel German. 2012. Contemporary peer review in action: Lessons from open source development. *IEEE software* 29, 6 (2012), 56–61.

[61] Peter C Rigby and Christian Bird. 2013. Convergent contemporary software peer review practices. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM, 202–212.

[62] Peter C Rigby, Daniel M German, Laura Cowen, and Margaret-Anne Storey. 2014. Peer review on open-source software projects: Parameters, statistical models, and theory. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 23, 4 (2014), 35.

[63] Peter C. Rigby, Seth Rogers, Sadruddin Saleem, Parth Suresh, Daniel Suskin, Patrick Riggs, Chandra Maddila, Nachiappan Nagappan, and Audris Mockus. 2025. Improving Code Reviewer Recommendation: Accuracy, Latency, Workload, and Bystanders. *ACM Trans. Softw. Eng. Methodol.* (May 2025). doi:10.1145/3736405 Just Accepted.

[64] P. C. Rigby, Y. C. Zhu, S. M. Donadelli, and A. Mockus. 2016. Quantifying and Mitigating Turnover-Induced Knowledge Loss: Case Studies of Chrome and a Project at Avaya. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. 1006–1016. doi:10.1145/2884781.2884851

[65] Guoping Rong, Yifan Zhang, Lanxin Yang, Fuli Zhang, Hongyu Kuang, and He Zhang. 2022. Modeling review history for reviewer recommendation: A hypergraph approach. In *Proceedings of the 44th international conference on software engineering*. 1381–1392.

[66] Christoffer Rosen, Ben Grawi, and Emad Shihab. 2015. Commit guru: analytics and risk prediction of software commits. In *Proceedings of the 2015 10th joint meeting on foundations of software engineering*. 966–969.

[67] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code Llama: Open Foundation Models for Code. arXiv:2308.12950 [cs.CL]

[68] Shade Ruangwan, Patanamon Thongtanunam, Akinori Ihara, and Kenichi Matsumoto. 2019. The impact of human factors on the participation decision of reviewers in modern code review. *Empirical Software Engineering* 24 (2019), 973–1016.

[69] 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: Software Engineering in Practice*. ACM, 181–190.

[70] Max Schäfer, Sarah Nadi, Aryaz Eghbali, and Frank Tip. 2023. An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation. arXiv:2302.06527 [cs.SE]

[71] Qianhua Shan, David Sukhdeo, Qianying Huang, Seth Rogers, Lawrence Chen, Elise Paradis, Peter C. Rigby, and Nachiappan Nagappan. 2022. Using nudges to accelerate code reviews at scale. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Singapore, Singapore) *(ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 472–482. doi:10.1145/3540250.3549104

[72] Emad Shihab, Ahmed E Hassan, Bram Adams, and Zhen Ming Jiang. 2012. An industrial study on the risk of software changes. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. 1–11.

[73] Anton Strand, Markus Gunnarson, Ricardo Britto, and Muhmmad Usman. 2020. Using a context-aware approach to recommend code reviewers: findings from an industrial case study. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*. 1–10.

[74] Patanamon Thongtanunam, Chakkrit Tantithamthavorn, Raula Gaikovina Kula, Norihiro Yoshida, Hajimu Iida, and Ken-ichi Matsumoto. 2015. Who should review my code? a file location-based code-reviewer recommendation approach for modern code review. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 141–150.

[75] Mahesh Kumar Thota, Francis H Shajin, P Rajesh, et al. 2020. Survey on software defect prediction techniques. *International Journal of Applied Science and Engineering* 17, 4 (2020), 331–344.

[76] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[77] Vinod Veeramachaneni, Pradeep Kumar Mallick, Subhashree Rout, and Piyush Kumar Pareek. 2025. Two-dimensional multi-deep CNNs for accurate and robust software defect detection: A performance-driven approach. In *2025 International Conference on Emerging Systems and Intelligent Computing (ESIC)*. IEEE, 734–739.

[78] Romi Satria Wahono. 2015. A systematic literature review of software defect prediction. *Journal of software engineering* 1, 1 (2015), 1–16.

[79] Xin Xia, David Lo, Xinyu Wang, and Xiaohu Yang. 2015. Who should review this change?: Putting text and file location analyses together for more accurate recommendations. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 261–270.

[80] Zhenglin Xia, Hailong Sun, Jing Jiang, Xu Wang, and Xudong Liu. 2017. A hybrid approach to code reviewer recommendation with collaborative filtering. In *2017 6th International Workshop on Software Mining (SoftwareMining)*. IEEE, 24–31.

[81] Xinqiang Xie, Xiaochun Yang, Bin Wang, and Qiang He. 2022. DevRec: Multi-Relationship Embedded Software Developer Recommendation. *IEEE Transactions on Software Engineering* 48, 11 (2022), 4357–4379. doi:10.1109/TSE.2021.3117590

[82] Kazuhiro Yamashita, Changyun Huang, Meiyappan Nagappan, Yasutaka Kamei, Audris Mockus, Ahmed E Hassan, and Naoyasu Ubayashi. 2016. Thresholds for size and complexity metrics: A case study from the perspective of defect density. In *2016 IEEE international conference on software quality, reliability and security (QRS)*. IEEE, 191–201.

[83] Haochao Ying, Liang Chen, Tingting Liang, and Jian Wu. 2016. Earec: leveraging expertise and authority for pull-request reviewer recommendation in github. In *Proceedings of the 3rd international workshop on crowdsourcing in software engineering*. 29–35.

[84] Yue Yu, Huaimin Wang, Gang Yin, and Tao Wang. 2016. Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment? *Information and Software Technology* 74 (2016), 204–218.

[85] Motahareh Bahrami Zanjani, Huzefa Kagdi, and Christian Bird. 2016. Automatically Recommending Peer Reviewers in Modern Code Review. *IEEE Trans. Softw. Eng.* 42, 6 (June 2016), 530–543. doi:10.1109/TSE.2015.2500238

[86] Jiyang Zhang, Chandra Maddila, Ram Bairi, Christian Bird, Ujjwal Raizada, Apoorva Agrawal, Yamini Jhawar, Kim Herzig, and Arie van Deursen. 2023. Using large-scale heterogeneous graph representation learning for code review recommendations at Microsoft. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 162–172.