Using Large Language Models to Support the Workflow of Differential Testing

Arun Krishna Vajjala[1], Ajay Krishna Vajjala[1], Carmen Badea[2], Christian Bird[2], Jade D'Souza[2], Robert DeLine[2], Mikhail Demyanyuk[2], Jason Entenmann[2], Nicole Forsgren[2], Aliaksandr Hramadski[2], Haris Mohammad[2], Sandeepan Sanyal[2], Oleg Surmachev[2], Thomas Zimmermann[3]

[1]George Mason University, [2]Microsoft Corp., [3]University of California, Irvine
{akrishn,akrish}@gmu.edu
{cabadea,cbird,jadedsouze,rdeline,midemyan,jentenmann,niforsgr,ahramadski,harismo,ssanyal,olegsu}@microsoft.com
tzimmer@uci.edu

Abstract

Many software development teams use differential testing as a quality gate in their release process. Differential testing—namely, comparing behavioral differences between a system in production and a system in test—is a laborious process to label changes as regressions, expected changes, or incidental changes (e.g. those due to nondeterminism or timing). This manual process involves inspecting large textual artifacts, like logs, pull requests, and team discussions, which suggests that Large Language Models (LLMs) could be helpful. In this paper, we engage with the team developing a central Azure service to understand their work practice for differential testing. We used a design probe method to elicit feedback about several ways to use LLMs to improve their work practice, including automatically labeling behavior differences and providing summaries of various artifacts and discussions. Release engineers on the team report that predicting a difference's label would save them effort, but they want an explicit rationale to improve their trust in the prediction; they found the generated summaries to be informative, if a bit wordy.

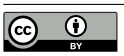
CCS Concepts

• **Software and its engineering → Software testing and debugging**.

Keywords

AI for SE, Differential Testing, Release Engineering, Developer Productivity, Large Language Models, Human-Computer Interaction

ACM Reference Format:
Arun Krishna Vajjala[1], Ajay Krishna Vajjala[1], Carmen Badea[2], Christian Bird[2], Jade D'Souza[2], Robert DeLine[2], Mikhail Demyanyuk[2], Jason Entenmann[2], Nicole Forsgren[2], Aliaksandr Hramadski[2], Haris Mohammad[2], Sandeepan Sanyal[2], Oleg Surmachev[2], Thomas Zimmermann[3] . 2025. Using Large Language Models to Support the Workflow of Differential Testing. In *33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion '25), June 23–28, 2025, Trondheim, Norway*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3696630.3728559



FSE Companion '25, Trondheim, Norway

ACM ISBN 979-8-4007-1276-0/2025/06
https://doi.org/10.1145/3696630.3728559

1 Introduction

The increasing complexity of software development and release processes has highlighted a challenge for large organizations: maintaining high software quality while improving developer productivity and workflows [4, 6, 7]. In fast-paced environments at companies such as Microsoft, where millions of users rely on seamless and reliable updates, software engineers often face overwhelming demands during the release phase. For engineers tasked with ensuring the reliability of new builds, traditional workflows can become a significant bottleneck, imposing cognitive burdens and inefficiencies that hinder productivity.

Developer productivity is not merely a technical concern but a business imperative[13, 26, 27]. In an industry where time-to-market and software reliability are imperative, inefficiencies in software engineer workflows directly impact operational costs, product quality, and customer satisfaction [6, 24, 31]. Tools that enable engineers to work more effectively can save organizations millions of dollars annually, reduce time-to-market for new features, and ensure smoother software rollouts [22]. Improving productivity in the release engineering phase of the software engineering life cycle, therefore, aligns with broader business objectives, enhancing organizational competitiveness in the software market.

Release engineering [2], a cornerstone of the software engineering lifecycle, often involves techniques like differential testing [11], where the behaviors of test and production builds are compared on identical data input to uncover potential regressions or inconsistencies [12, 19, 33]. While effective at identifying issues, these methods rely heavily on manual workflows, requiring engineers to investigate extensive logs and differential information [3, 16, 17]. In particular, software engineers need to distinguish regressions from expected behavior differences due to feature enhancements and bug fixes. This process is complicated by the presence of inconsequential differences ("noise") caused by asynchrony, timing differences, non-determinism, etc. Deciding whether a behavioral difference is expected, a regression or noise may require inspecting team artifacts beyond the logs themselves, like recent code changes, bug databases, and team communication channels. To ensure release quality, this process of categorizing differences is repeated for each new build.

The overall workflow involves the inspection of many textual artifacts, which suggests that Large Language Models (LLMs) could be helpful in providing support and automation for aspects of this work. Historically, release engineering tools have been designed to

provide data and rely on engineers for interpretation, prioritization, and action [8, 10]. While this model sufficed in simpler development environments, modern software systems demand tools that do more than just display information. Intelligent ML and AI tools can automate, generate, and analyze complex information to provide advanced insights, enabling software engineers to identify critical issues and opportunities more effectively [3, 8, 26]. Proactive AI represents the next stage in this evolution, offering tools that not only identify patterns and automate analysis but also provide actionable insights and recommendations tailored to specific workflows, both making the process easier for experienced engineers and limiting the learning curve [29].

To explore the potential of LLMs to improve the work practice of differential testing, we engaged with the team responsible for a critical service within Microsoft Azure. This team conducts differential testing as a routine part of their release process and have substantially invested in making this workflow productive. They established a work regimen in which roughly 30 members of the team take shifts to categorize behavioral differences, during which they are called **on-call engineers (OCEs)**; the OCEs are spread across geo-locations to keep the work going around the clock. The OCEs use a tool (**DiffViewer**) specifically designed to allow them to browse, inspect, and categorize behavior differences in detail (Figure 1 Left Side). Even with this level of investment, the team finds differential testing laborious, consuming resources that might otherwise go toward other engineering efforts.

To address these limitations, we propose enhancements for the DiffViewer tool, in an attempt to enhance release engineering workflows with AI, showcasing a novel application of existing methods such as Large Language Models (LLMs) and clustering to push forward differential testing and elevate the effectiveness of developer tools in this space. To the best of our knowledge, this is the first piece of work where LLMs and AI automation techniques were used in differential testing tooling for software engineers. By integrating AI capabilities, such as LLM-based diff label prediction and automated summarization, alongside thoughtful user interface enhancements, we can transform how software engineers interact with release engineering tools like the DiffViewer, further providing a more intelligent way of streamlining their workflow. The redesigned DiffViewer introduces features like machine learning-based clustering, prioritization mechanisms, and a block-based layout with progress tracking—enabling software engineers to focus on critical tasks while automating repetitive ones. This shift from manual to proactive workflows not only reduces cognitive load but also enhances the development experience, empowering software engineers to deliver high-quality software with greater efficiency and satisfaction. This work makes the following contributions to address previous challenges:

- We leverage Large Language Models (LLMs) to automate differential testing processes, generate contextualized summaries, and provide actionable insights.
- We incorporate machine learning-based clustering and prioritization mechanisms to enable OCEs to focus on related and important issues while serving as an organizational tool.
- We prototype the evolution of a prior tool from a manual workflow tool to a more guided approach, offering a user-centric enhancement to release engineering tooling while providing valuable insights.

2 Approach

This paper provides a series of enhancements to the original DiffViewer tool used internally at Microsoft.

2.1 Original DiffViewer

The raw differences between the metadata of the released and candidate builds can be quite large and include many redundant occurrences of the same difference due to loops, repeated invocations, and other iterative behavior. To remove these redundancies, the DiffViewer bins individual log differences by their program locations, keeping a count of occurrences for each. The team uses the term **diff** to refer to a binned log difference with a count. The DiffViewer tool shows all diffs organized as rows in a table, sorted from highest to lowest count. The left side of Figure 1 depicts the original DiffViewer, where rows of diffs are displayed with basic features such as counts, owners, categories, and a comment section. The right side of Figure 1 shows the updated version, which enhances the process with a more visually organized interface, AI-generated summaries, and streamlined tools for categorization and ownership assignment. In a typical on-call session, an OCE works through the Diffs from top to bottom with the goal of categorizing the Diff (as regression, expected behavior, or noise) and assigning an engineer to be responsible for that Diff (the "owner"). Each Diff also has a comment section where team members can add information, such as justifications for suggested categories.

2.2 Redesign Through Developer Insights

Given the nature of the DiffViewer, gathering developer insights was crucial to understand its current usage and identify potential improvements. To achieve this, we conducted a series of interviews with four software engineers across varying levels of experience with the DiffViewer. This approach ensured that we captured perspectives from new, experienced, and senior engineers regarding the process of labeling diffs within the tool.

We reached out to software engineers based on the number of Diffs they had labeled, ensuring a balanced representation of experience levels by inviting individuals who had labeled between 100 and 1,000 Diffs, 1,000 and 3,000 Diffs, and more than 3,000 Diffs. This approach provided insights from both novice and experienced software engineers, avoiding bias toward any particular skill level. The interviews, which included live demonstrations, captured both qualitative and observational data as participants labeled Diffs while explaining their thought processes. Emphasis was placed on understanding their UI interactions, including which columns or elements were most frequently referenced, to identify the most valuable components, especially given the redundant processes in the original DiffViewer.

During the live demonstration sessions, we observed common behaviors such as examining error codes, referencing categories and descriptions, and comparing details between test and production environments, which helped us understand how engineers

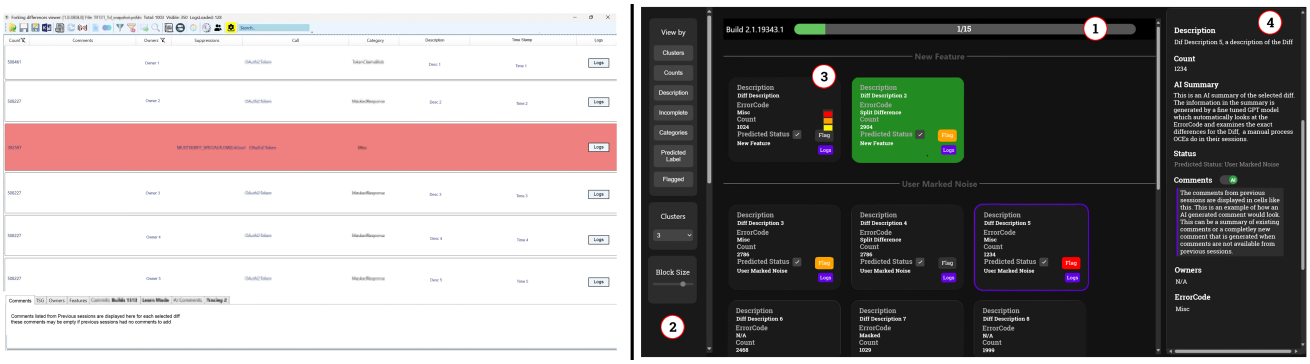


Figure 1: Comparison of the original diff viewer (left) and the new version (right). The new version introduces a more structured layout, improved visual indicators, and AI-generated summaries to enhance usability and streamline workflow.

collected and analyzed information to label diffs accurately. We also observed which parts of the UI participants relied on most, providing insights into their workflows and decision-making. Additionally, we gathered direct feedback on the DiffViewer's design and functionality, with many participants suggesting features to reduce cognitive load and manual effort, such as automation for information retrieval, predictive tools for error categorization, and organizational improvements to prioritize critical information.

The interview process revealed how software engineers used the DiffViewer, highlighting pain points, valuable features, and opportunities for improvement. By combining insights from both direct observations and feedback, we gained a comprehensive understanding of how to redesign the DiffViewer to provide a more intuitive, streamlined experience for developers at all levels. Key areas for enhancement included improved automation, streamlined workflows, and better tools for organizing and prioritizing information. Based on these findings, we created a prototype embodying ideas to make the DiffViewer more efficient, intuitive, and developer-friendly. The goal of the prototype is to elicit feedback from OCEs and team leaders.

2.3 Design Probe

The OCEs' work is both critical to the Azure service's quality and full of time pressure. Even with its limitations, the existing DiffViewer tool is reliable and familiar to OCEs. To introduce a replacement tool or even to change the existing features requires retraining the OCEs and taking a risk that the change is more effective. Therefore, to get feedback while not disturbing the team's ongoing work, we chose a design probe methodology [15]. We designed a prototype to demonstrate several possible enhancements to the existing work practice, with the goal of eliciting insights, reflections, and feedback from OCEs.

The redesigned DiffViewer interface introduces a block-based layout that prioritizes clarity, ease of access, and efficiency. Due to the sensitive nature of the information processed by the tool, certain details have been blurred out or replaced with placeholder text in the images presented here. However, the interface remains identical in structure and functionality for real-world users.

The redesigned DiffViewer UI (Figure 1 Right Side) is divided into distinct, intuitive panels. The top panel (1) displays the build number and progress bar, allowing users to track their progress within each session on the latest build. This feature gamifies the workflow while ensuring developers remain informed about the overall status of their work. In the left panel (2), users can efficiently organize their tasks, with options to filter, group, and sort Diffs based on various criteria. This panel enables quick navigation to specific areas of interest, significantly reducing the time required to locate and address key issues. The central block (3) forms the core of the UI, presenting the most important and immediate information to OCEs. Each block contains comprehensive details about a Diff, including predicted labels, descriptions, and log access, ensuring that all relevant information is readily accessible within the block. OCEs can also use this area to flag items for further review or use the checkmark to confirm the predicted label as their desired label for the diff. An example of this is shown in the image where the Green block is a labeled Diff, for New Feature. This labeling is also reflected in the Progress Bar (1) at the top where 1 of the 15 Diffs are labeled. On the right side panel (4), additional insights and summaries generated by the integrated AI are displayed. This feature provides contextual, AI-driven summaries and patterns extracted from the logs as well as comment generation and summary, providing more information about the Diff to the OCEs. By leveraging machine learning to offer proactive insights, the redesigned DiffViewer minimizes manual effort and enhances decision-making.

Overall, this prototype exemplifies a user-centric approach tailored to the high-stakes context of Microsoft's security initiatives within Azure. By blending automation, organization, and intuitive design, the tool empowers OCEs to work more efficiently and effectively.

2.4 LLM-Based Diff Label Prediction

A central feature of the redesigned DiffViewer is its capability to automate diff labeling through the use of Large Language Models (LLMs). This builds on prior work by Krishna Vajjala et al. [32], which presented a robust approach for utilizing LLMs to classify differences between test and production builds. Fine-tuned on historical datasets with thousands of Diffs, the model achieves an overall accuracy of 97.38%, with low false-positive and false-negative rates, ensuring reliable predictions across categories such as *UserMarkedNoise*, *NewFeature*, and *Regression*. The fine-tuned model

used a balanced set of data between the three labels, to predict labels. By integrating this feature into the tool, we address one of the most time-consuming and cognitively demanding aspects of the Diff analysis workflow, streamlining the process for OCEs and enabling them to focus on higher-priority tasks.

The LLM-based Diff labeler is integrated into the redesigned DiffViewer's UI, where predictions are displayed alongside confidence scores and supporting metadata to enhance transparency and usability. The tool makes it easy for OCEs to confirm predictions with a single click on a checkmark next to the suggested label, automatically labeling the Diff. This combination of high accuracy, reliability, and user-friendly design significantly reduces manual effort and ensures consistent labeling, transforming the diff analysis workflow into a more efficient and intuitive process.

Within the tool, Diff label prediction integrates with the grouping feature in Section 2.7, allowing OCEs to sort and organize Diffs by predicted labels. This enables quick identification of critical regressions while de-prioritizing noise, streamlining the workflow and focusing attention on high-impact issues. In the UI, predictions are displayed with a checkmark next to them, allowing OCEs to easily confirm a prediction with a single click, automatically labeling the diff and further reducing manual effort. This integration motivates the tool's philosophy of combining intelligent automation with user-centric design, significantly reducing cognitive load during on-call sessions.

By incorporating LLM-based Diff label prediction into the broader DiffViewer framework, the tool transforms the Diff analysis workflow from a manual process into a proactive and guided experience.

2.5 Automation of Information Analysis and Summarization

This feature streamlines differential testing by automating the extraction and summarization of log information OCEs use to label Diffs. During interviews, we observed OCEs following heuristic patterns: identifying the Diff Category label, locating this label in the differential view, and analyzing log differences to make decisions. To simplify this, we automated these steps by programmatically identifying labels in the Diff Category and extracting the key differences from the logs. This reduces the need for manual searching and comparison, significantly lowering developers' cognitive load and saving time.

Building on this automation, we enhance the workflow by using the extracted log differences and category labels as context for our fine-tuned LLM described in Section 2.4. The LLM generates a high-context summary of the Diff, providing OCEs with both the extracted differences and an intelligent summary of key distinctions. This eliminates the need for OCEs to manually interpret the logs, enabling them to review the summary and make informed decisions. By combining automation with an explanation of the predicted label, we enhance explainability and allow OCEs to confidently confirm the labeling of each diff. This approach exemplifies proactive AI by automating redundant, time-consuming tasks and empowering developers to focus on higher-level decision-making, moving beyond traditional chat-based AI usage and creating a more efficient, developer-friendly workflow.

2.6 Comment Summarization and Generation

This feature streamlines Diff labeling by generating concise summaries of prior comments and creating new ones using a fine-tuned LLM. OCEs in on-call sessions often reference existing comments for context, however, reviewing dozens of comments can be time-consuming. The LLM processes Diff logs and associated comments to produce summaries, prioritizing recent and relevant insights. A toggle switch allows developers to view either the raw comments or the summary, depending on their needs. For Diffs without prior comments, the LLM generates new, context-aware suggestions based on patterns from historical data, providing more information for Diffs without comments.

2.7 Clustering Options for Similar Diffs

This section discusses how we introduced a novel application of the K-means clustering algorithm to organize and work with Diffs in the context of differential testing. Leveraging insights from our developer interviews, we designed a system that allows developers to cluster and sort Diffs dynamically, enabling them to work more efficiently based on their preferences and priorities. Below, we outline the clustering methodology, UI design considerations, and prioritization strategies.

2.7.1 Clustering Methodology. Through our interviews, we identified two key attributes of diffs that naturally lend themselves to clustering: Diff Categories and Predicted Labels. By default, Diffs are grouped into six groups based on the six distinct, pre-existing Diff Categories. Similarly, Diffs can also be grouped into three groups based on the three possible predicted labels.

For more granular control, we allow clustering based on the description of the Diff. In this case, we generate embeddings for the Diff descriptions using OpenAI embeddings and perform k-means clustering. OCEs can specify the number of clusters for description-based clustering, ranging from 3 to 10 clusters, giving them the flexibility to adapt the organization of their diffs to their needs.

This combination of fixed clusters for categories and labels, along with dynamic clusters based on description embeddings, provides a robust and flexible clustering framework. To ensure real-time performance, we cache the embeddings for the Diff descriptions within the data of each Diff, enabling quick computations even with up to 100 Diffs in a session, as observed from our interviews and testing.

2.7.2 Dynamic UI Design for Clustering. The UI incorporates clear cluster separators that visually distinguish clusters with a dividing line. Each cluster separator includes a concise 5-7 word summary of the cluster, recalculated dynamically whenever a new clustering request is made. This summary provides OCEs with an overview of the cluster contents, helping them quickly identify the type of diffs grouped together.

The UI also allows developers to toggle between grouping or clustering by categories, labels, or descriptions, giving them control over how they organize their Diffs. For description-based clustering, developers can dynamically adjust the number of clusters in real time, further enhancing flexibility and usability.

2.7.3 Prioritization Within Clusters. To account for the importance of diffs, we implemented a prioritization mechanism based on the

"count" value of each Diff, which signifies how frequently a Diff occurs. Higher counts represent higher priority Diffs. Regardless of the clustering method chosen, Diffs within each cluster are always ordered by their counts, ensuring that developers can address the most critical Diffs first.

The clustering options provide OCEs with the flexibility to organize Diffs dynamically based on categories, predicted labels, or description similarity. By combining fixed groups for categories and labels with dynamic embedding-based clusters for descriptions, the system balances structure and adaptability. With real-time performance ensured through embedding caching and prioritization mechanisms, developers can work efficiently and focus on the most critical tasks.

2.8 Flagging for Organization

This feature introduces a flagging system for prioritizing, following up, or analyzing Diffs, addressing a major gap in the previous workflow. Previously, OCEs lacked an efficient way to revisit specific diffs, often resorting to manually sifting through lists.

To address this, we implemented a flagging functionality that allows developers to mark Diffs with one of three colored flags: Red, Orange, or Yellow. Importantly, these colors do not have predefined meanings, giving OCEs the freedom to use the flags in a way that aligns with their individual workflows and organizational preferences. To enhance usability, we also provide a filtering feature that allows OCEs to view only flagged Diffs, enabling them to focus exclusively on Diffs they marked as important. The flag colors remain visible even outside the filtered view, ensuring that flagged Diffs are easily identifiable when working with the full list of Diffs. Interviews revealed that revisiting unresolved Diffs was a key pain point. This flexible system empowers OCEs to customize their workflow, fostering control and improving organization, making the redesigned DiffViewer more intuitive and user-friendly.
.

2.9 Progress Bar

The progress bar is a simple yet essential addition to the redesigned DiffViewer, providing OCEs with a clear visual representation of their progress during a labeling session. This feature allows developers to easily see the total number of Diffs in a session and track how many have been labeled.

By offering incremental progress updates, the progress bar transforms the labeling process into a more visual and target-driven experience. This clear progress indication not only helps OCEs stay organized but also enhances the overall user experience by fostering a sense of accomplishment and reducing the cognitive load of managing their workflow.

2.10 Complete/Incomplete Status

This simple yet impactful feature enhances workflow organization by introducing a clear distinction between labeled and unlabeled Diffs. The original DiffViewer presented a cluttered view where all Diffs, regardless of their status, remained visible, often leading to visual overwhelm. To address this, we implemented an inbox-style view that focuses exclusively on unlabeled diffs.

With this feature, once a Diff is labeled, it is automatically removed from the view, leaving only the incomplete Diffs. This streamlined approach reduces visual noise, helps OCEs stay focused, and ensures a cleaner, more organized workflow.

2.11 Priority-Based Ordering and Reduced Mouse Travel

The Priority-Based Ordering feature, inspired by the original DiffViewer, allows OCEs to view all of the Diffs in order of priority, ensuring critical issues are addressed promptly. In typical on-call sessions, labeling can take 5–6 hours, as OCEs methodically identify discrepancies between production and test logs. By prioritizing the most persistent errors early in the session, this feature enhances the efficiency and impact of the labeling process.

Building on observations from OCE interviews, the Reduced Mouse Travel feature addresses inefficiencies caused by scattered information across the screen. A redesigned visual layout consolidates all necessary actions and information within a single block for each diff. When an OCE selects a Diff block, related information—such as category, summary, and descriptions—appears on the right side of the screen. Additionally, a label button is integrated directly into the diff block, enabling OCEs to label the Diff without navigating elsewhere. This centralized design minimizes screen traversal, significantly improving workflow efficiency and reducing mouse travel.

2.12 Preserving Core Features

While prototyping several new features, we ensured that the redesigned DiffViewer retained key elements from the original version that OCEs relied upon. Core features such as side-by-side production and test log comparisons and Diff labeling remain central to the tool. These features were essential to the workflow and heavily utilized by OCEs, making them critical to preserve. For example, side-by-side log comparisons are now easier to access within the block-based UI, and labeling Diffs is streamlined with the addition of label buttons directly on each block. We ensured that the proactive AI integration and automation features were designed to seamlessly align with the existing workflow, allowing developers to adopt these enhancements without disrupting their familiarity with the original tooling. This approach ensures that the DiffViewer builds upon its strengths while addressing the pain points identified during developer interviews, providing both continuity and enhanced usability.

3 Design Probe Study

The prototype embodies a number of potential improvements to the work practice, with the goal of eliciting feedback from OCEs about which ideas to prioritize for further investment. Because the OCEs' work is both critical and laborious, it would not be reasonable to ask them to conduct the work using a prototype during their on-call sessions nor to repeat the work during their off times. Instead, we use a design probe (also called a technology probe) method, which combines multiple goals: "the social science goal of understanding the needs and desires of users in a real-world setting, the engineering goal of field-testing the technology, and the design goal of inspiring users and researchers to think about

new technologies" [15]. In particular, we used the prototype as a shared artifact during a series of interviews with OCEs, to elicit feedback about their preferences and priorities.

3.1 Participants

The study had a participation rate of 17%, with 5 out of 30 invited OCEs electing to participate. All participants were experienced with the Diff labeling workflow, ensuring relevant and practical feedback was obtained. OCEs work in shifts across two teams, one in Ireland and one in Redmond, Washington, USA. Four of the participants were from the Redmond team and one from the Ireland team. We conducted the study during the OCEs off-call times, in deference to their critical role. Sessions were 30–45 minutes, for which participants were compensated $200 USD. All participants signed a consent form to allow transcripts of their sessions to be used for this study.

3.2 Interview Process

The evaluation of the redesigned DiffViewer involved a demonstrative interview process to gather qualitative feedback from participants. The purpose of these interviews was to assess the usability and effectiveness of the new features while ensuring that authentic feedback drove the analysis and future improvements.

Each session followed a structured yet flexible protocol to guide discussions and encourage open-ended feedback. Sessions began with an introduction to the prototype, clarifying that it was not intended to replace the existing tool but to showcase new features designed to enhance the diff labeling process.

Participants were guided through the prototype's core layout and workflow, which included four main areas: a central tile-based view displaying all Diffs, a progress bar tracking completion, a control panel for managing displayed information, and a detail panel providing comprehensive insights into selected diffs. Demonstrations highlighted key features such as clustering by similarity, flagging, AI-based label predictions, and AI-generated summaries of logs and comments. For each feature, participants were asked targeted questions about its functionality, usefulness, and alignment with their workflow. These questions aimed to assess the improvements from the participants' perspective, identify features that resonated with their needs, and determine their preferences. As the prototype showcased potential features, the goal was to understand which elements participants envisioned incorporating into their daily tasks.

The demonstrative nature of the process allowed participants to engage with the prototype in a guided manner. By visually showcasing features and prompting discussions, the study encouraged participants to share their likes, dislikes, and suggestions for improvement. This approach ensured that feedback was focused yet open-ended, providing valuable insights into both the strengths and limitations of the redesigned DiffViewer.

The interview process concluded with a short questionnaire, where participants rated the features and provided additional feedback. This combination of guided demonstrations and structured surveys offered a comprehensive understanding of the tool's impact and areas for refinement.

3.3 Protocol Overview

The study protocol was designed to comprehensively evaluate the redesigned DiffViewer tool while fostering an open environment for candid and constructive feedback. The primary goal was to assess the usability, functionality, and impact of the new features in the redesigned DiffViewer on the workflow of OCEs. The protocol balanced standardized data collection with adaptability to capture participants' unique insights and experiences.

Each session began with a brief introduction to the research objectives, highlighting the experimental nature of the prototype and its role in exploring innovations for diff analysis. Participants were assured that the tool was not intended to replace their current systems but to gather feedback on potential features. The introduction also outlined the session structure and clarified that the prototype was solely for demonstration purposes, with no impact on their existing workflows or databases.

Following the introduction, participants were guided through the core features of the prototyped DiffViewer tool, with an emphasis on its primary functionalities. They learned to select specific builds and navigate the interface efficiently using a preloaded build, enabling a focused exploration of the tool's features. Machine learning-based clustering was highlighted as a means of grouping similar Diffs, streamlining labeling tasks, and reducing cognitive load, with participants providing feedback on this approach compared to traditional sorting methods. Flexible viewing modes, such as sorting by count, category, or incompleteness, were demonstrated alongside dynamic filtering and pivoting capabilities. AI-powered features, including label predictions and concise Diff summaries, were introduced to save time and reduce manual effort, and participants evaluated their accuracy and utility. Interactivity was encouraged, allowing participants to experiment with the tool and provide feedback on its design and functionality.

Throughout the session, participants shared their impressions of the tool and its integration into their workflows. Discussions focused on clustering, AI predictions, and user interface design, providing insights into the tool's strengths and potential areas for improvement.

The study concluded with a questionnaire to collect quantitative ratings and qualitative feedback on the DiffViewer's features. Participants were asked to rate their overall satisfaction, comment on the tool's potential integration into their workflows, and suggest any additional functionalities they deemed necessary. This final step ensured that the study captured a holistic perspective on the tool's capabilities and future development opportunities.

The protocol combined structured demonstrations, interactive exploration, and open-ended discussions for a thorough evaluation, ensuring authentic, actionable insights reflective of participants' real-world experiences.

3.4 Survey

The survey was conducted immediately following the DiffViewer tool demonstration to collect feedback on OCEs' experiences, impressions, and potential areas for improvement. It featured both quantitative and qualitative questions, focusing on usability, functionality, and the tool's impact on workflows. Quantitative data

offered valuable insights for evaluating the tool's effectiveness and identifying priorities for further development.

Survey questions were designed to address specific objectives. OCEs were asked about the ease of navigation and intuitiveness of the interface to assess whether the design met user expectations. Questions on functionality examined the relevance and accuracy of the tool's features, providing insights into how well it supports essential tasks. Feedback on workflow impact explored how the tool influenced productivity and task efficiency. Open-ended questions invited participants to identify shortcomings or suggest additional features for enhancement. Overall satisfaction was measured using a Likert scale, offering a standardized metric for evaluation.

Table 1: Challenges and Features Evaluated in the DiffViewer Study

Survey Category	Details
Biggest Challenges: Ranked from most challenging to least challenging	Difficulty finding a specific diff.
	Finding a previously worked-on diff.
	Jumping between screens (e.g., logs, code) when diagnosing a diff.
	Assigning a diff to the right owner.
	Re-establishing context when moving between diffs.
	Writing summaries when assigning diffs to owners.
	Time required to manually label predictable diffs.
	Estimating work progress (remaining or completed).
	Other unspecified challenges.
Feature Evaluation: 5 point Likert scale from Not-Likely to Very-Likely to use a feature	Label predictions for review.
	Progress bar.
	AI summaries in the detail panel.
	Flagging diffs.
	Alternative views (e.g., by count, cluster).
	Clustering diffs.
	Tile layout for diffs.
	Detail panel on the right.

In designing the questionnaire, we used two complementary approaches—ranking and a 5-point Likert scale—to gather insights about OCEs' workflows and the DiffViewer tool (Table 1). The "Biggest Challenges" section asked participants to rank challenges from most to least impactful, prioritizing areas where OCEs face the most friction. This approach forces participants to identify critical obstacles, such as whether locating specific diffs or re-establishing context poses a greater challenge, providing clear, actionable priorities for improvement.

The "Feature Evaluation" section employed a 5-point Likert scale to assess the perceived usefulness and likelihood of adoption of specific redesigned DiffViewer features. This method allowed us to capture detailed feedback on various features, from label predictions to clustering and alternative views, revealing the features participants are most likely to use and those requiring refinement. To encourage participation while collecting actionable feedback, the questionnaire was kept concise. By balancing quantitative rankings with qualitative Likert-scale assessments, the study ensured a holistic understanding of the tool's impact on OCE workflows.

3.5 Artifacts Collected

The evaluation process generated two primary types of artifacts, both of which provide valuable insights into the usability and functionality of the DiffViewer tool:

3.5.1 Transcripts: Transcripts from interview sessions captured detailed interactions and participant feedback on the tool. They provide qualitative data on features, workflow impact, and suggestions for improvement, along with spontaneous comments that reveal participant impressions and help identify recurring themes or unique insights.

3.5.2 Survey Responses: Post-session surveys collected structured feedback, including feature preferences, usability impressions, and prioritized functionalities. These responses combine quantitative ratings (e.g., Likert scale scores) with qualitative insights from open-ended questions, enabling a balanced analysis of the tool's strengths, weaknesses, and areas for improvement.

By combining these two types of artifacts, the evaluation ensures a comprehensive understanding of participant feedback. The transcripts offer depth and context through rich qualitative insights, while the survey responses provide structured data to support quantitative analysis. These artifacts collectively serve as the foundation for evaluating the DiffViewer tool's impact and guiding its iterative refinement.

4 Results

4.1 Ranking of Challenges

As part of our evaluation, we conducted a structured survey to better understand the challenges OCEs face in the Diff analysis workflow. This survey provided participants the opportunity to rank and prioritize the difficulties they encounter, offering a clear view of the most significant pain points in the process. The ranking information across the participants can be seen in Figure 2.

The results of our structured survey highlight significant challenges OCEs face during on-call workflows, with the most frequently cited issues being the difficulty of jumping between screens, finding specific Diffs or those previously worked on, and manually labeling predictable diffs. These challenges reflect the fragmented and time-consuming nature of the original DiffViewer workflow, requiring OCEs to expend significant effort navigating multiple screens, interpreting logs, and revisiting prior work.

To address these issues, the redesigned DiffViewer introduces key improvements that significantly enhance the Diff analysis process. A consolidated block-based UI, combined with various filtering and display options, eliminates the need to jump between screens by presenting all relevant information in a single, intuitive layout. The integration of LLM-powered predictive Diff labeling further reduces the burden of manually labeling predictable Diffs, while intelligent clustering and flagging features streamline the organization and revisiting of specific Diffs. Additionally, the tool automates aspects of information extraction and summarization, addressing investigative challenges by providing actionable insights directly

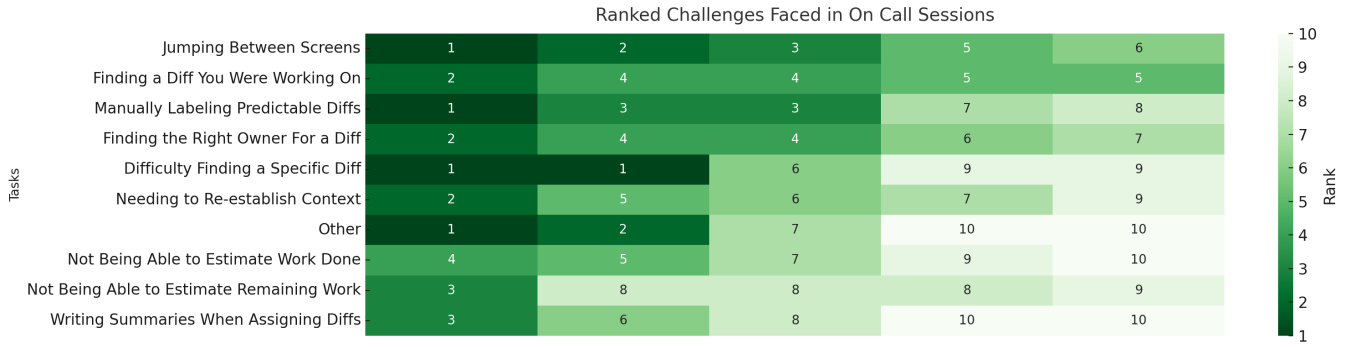


Figure 2: Ranked Challenges Faced in On-Call Sessions

within the interface. Features such as comment summarization and clustering also enhance communication and collaboration, enabling teams to seamlessly share and act on diff-related information.

Beyond addressing major challenges, the redesigned DiffViewer incorporates improvements targeting less critical yet impactful aspects of the workflow. For example, the introduction of a progress bar and an inbox-style view for incomplete Diffs enhances organization and gives OCEs a clear sense of progress during labeling sessions. Features like priority-based ordering, reduced mouse travel, and streamlined UI layouts tackle usability concerns, minimizing frustration during repetitive tasks.

By addressing both critical and secondary challenges, the redesigned DiffViewer delivers a smoother, more intuitive user experience that supports OCEs at every stage of the Diff analysis process, ultimately improving efficiency and reducing cognitive load.

Participants who reported "other" as a challenge mentioned the complexity of transferring Diffs, determining whether prior feedback could be reused with a new build, the investigative nature of analyzing code outside the tool, and the lack of efficient mechanisms for sharing Diff-related information across teams. Though the redesigned DiffViewer is not equipped with communication portals, we address the issue of context for Diffs via generative comments and summaries to aid the OCEs in making their decisions.

These results validate the redesigned DiffViewer and highlight key challenges OCEs face. By addressing these pain points, the tool improves efficiency, reduces cognitive load, and demonstrates strong alignment with real-world needs.

4.2 Feature Evaluation

Participants rated features of the prototype DiffViewer using a 5-point Likert scale, assessing their usability, relevance, and likelihood of adoption. The results provide a ranking of features, highlighting which enhancements appealed most to OCEs and why. The results of the findings can be seen in Figure 3. These findings also underscore the core motivation of this paper: leveraging AI and automation to address release engineering-specific tasks. The results reveal that software engineers want and need more intelligent tools to streamline workflows in the release engineering phase, and the features introduced in this study contribute to improved productivity and efficiency.

The feature allowing for alternative views, such as grouping and clustering Diffs by count, description, or cluster, emerged as the most likely to be adopted, receiving overwhelmingly positive feedback from participants. This reflects OCEs' strong preference for flexibility and customization in organizing and viewing Diffs, as these options enable them to adapt the tool to their specific needs. By offering multiple ways to cluster and sort Diffs, this feature directly enhances workflow efficiency and adaptability.

Similarly, the ability to flag Diffs was highly rated by OCEs, who found it particularly useful for marking critical items for follow-up or prioritization. The ability to filter flagged diffs and visually distinguish them in the interface aligns with OCEs' need for better task organization during on-call sessions, addressing the challenge of efficiently revisiting and managing specific Diffs.

The tile-based layout for presenting diffs also received high praise, with OCEs appreciating the streamlined and consolidated presentation of information. By reducing the cognitive load associated with navigating between screens and providing all relevant details in a single, clear layout, this feature effectively addresses one of the core challenges identified in the survey and interviews.

The feature of Diff label predictions garnered significant interest among OCEs, with many acknowledging its potential to reduce manual effort and streamline the labeling process. However, some participants raised concerns about accuracy and explainability, emphasizing the need for transparency in AI-driven tools, providing an opportunity to build trust in predictive systems by improving accuracy and providing clear justifications for predictions, which would facilitate broader adoption and seamless integration into workflows.

AI-generated summaries in the detail panel and the repositioned detail panel on the right also received positive feedback, though slightly lower than other features. OCEs appreciated the summaries for their ability to reduce manual effort in interpreting logs and synthesizing information, while the repositioned detail panel improved the clarity and accessibility of key details. Together, these features highlight the value of intuitive UI design and intelligent automation in streamlining the analysis process.

The progress bar, while receiving moderate ratings, was noted for its utility in tracking task completion during lengthy labeling sessions. Although it does not address a primary pain point, the

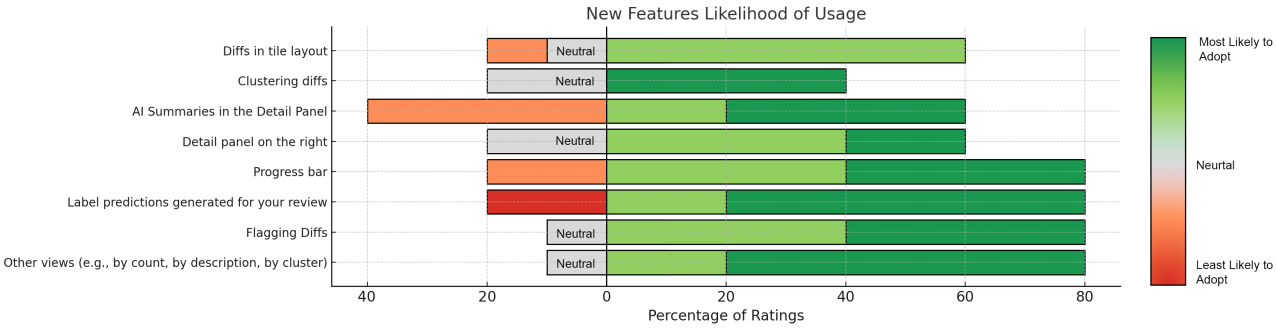


Figure 3: Survey of Likelihood of OCEs to Adobt New Features

progress bar contributes to a more structured and engaging workflow by providing a sense of progress and accomplishment.

In summary, the evaluation highlights a preference for features that offer flexibility, organization, and streamlined workflows, such as other views, flagging diffs, and the tile-based layout. These features significantly improve the efficiency and adaptability of the diff analysis process. By addressing both core challenges and secondary concerns, the redesigned DiffViewer demonstrates the potential of AI and automation in improving software engineering workflows and advancing the state of release engineering tools.

4.3 Qualitative Feedback Analysis

Qualitative feedback collected through open-ended survey responses and interviews provided valuable insights into user preferences, trust in AI, and areas for improvement. The key themes that emerged are outlined below:

Usability. Participants praised the intuitive layout of the redesigned DiffViewer, particularly the grouping feature. Grouping was noted as a significant improvement over the traditional linear workflow, reducing cognitive effort and minimizing "bouncing around" between unrelated tasks. One participant highlighted, *"It would probably decrease the chance of us bouncing around with the same issue...this is a good, good approach."* Another remarked on the utility of pivoting views, saying, *"Having different pivots makes sense... it's a newer way to organize the data"*, demonstrating the need for flexibility in exploring diffs.

However, there was a recurring suggestion to add additional filtering options to complement clustering and pivoting. For example, a participant requested the ability to filter based on team ownership: *"It would be very useful to filter based on teams or areas we own...this would help us focus only on what's relevant."*

AI Trust and Reliability. While participants found AI predictions generally useful, many expressed a desire for greater transparency in how predictions were generated. A participant articulated this sentiment clearly: *"I need to know what data led to the prediction...without that, I'd still have to do all the investigation myself."*

The concept of supporting metadata or rationale was a recurring request. As one OCE put it, *"If it told me the relevant code changes or logs that led to the label, it would save so much time."* That is, the rationale should not just be the logic behind the prediction, but a set of links to relevant artifacts to allow an OCE to quickly double-check or overrule the prediction. Another emphasized the need to build trust gradually: *"As I use the tool more and see that my results match its predictions, I'll grow to trust it more."*

Feature Suggestions. Several OCEs suggested features, emphasizing the value of clustering and predictive labeling. One participant proposed incorporating common errors into the AI logic, such as automatically marking known caching issues as low priority for quick review. Another suggestion was to allow the direct transfer of diffs between teams within the tool, reducing back-and-forth communication. Participants also requested a richer display of metadata in the detail panel to improve decision-making and workflow efficiency.

These insights support the motivations and approach of this paper, showing that OCEs seek intelligent tools that enhance productivity while maintaining transparency and flexibility. The feedback has directly influenced the features of the DiffViewer, addressing existing challenges and emerging needs in release engineering.

4.4 Comparison of AI/ML vs Non-AI/ML Enhancements Based on User Feedback

The redesigned DiffViewer introduced both AI-driven and non-AI-driven features, with participants providing detailed feedback on each. This section compares their perceived benefits, limitations, and overall impact on the workflow.

AI-driven enhancements were praised for their time-saving potential, particularly predictive diff labeling, which automated repetitive tasks and allowed OCEs to focus on higher-priority work. AI-generated summaries also reduced cognitive load by providing immediate context for logs, with one participant saying, *"The AI summaries are a great starting point—I don't have to dig through logs just to understand what's happening."* The intelligent clustering feature, which grouped related diffs, was valued for improving prioritization and managing large data volumes. However, concerns about accuracy and explainability of AI-driven features were frequently mentioned. Transparency was crucial for trust, with one participant stating, *"I trust the predictions more when I understand how they were made and what data was used."* These concerns highlight the need for clear justifications behind AI predictions and outputs.

In contrast, non-AI enhancements were appreciated for their reliability and simplicity. The consolidated block-based UI streamlined navigation, while the progress bar and inbox-style views helped track tasks and organize labeling sessions. Features like flagging and priority-based ordering improved task management. Overall, AI-driven features were effective in reducing effort and accelerating workflows but required trust-building through better explainability. Non-AI enhancements were seen as intuitive and immediately impactful, addressing key usability pain points.

Participants favored hybrid solutions that integrated AI-driven functionality with usability improvements. For example, the clustering feature, which combined machine learning for dynamic grouping with a user-friendly interface, was seen as a best-of-both-worlds solution. However, some participants hesitated to rely solely on AI for critical tasks. Together, each set of features complemented each other, delivering a balanced solution that met diverse OCE needs and encouraged broader adoption of AI-driven workflows.

5 Related Works

Differential testing is a key technique in software engineering that compares outputs from different system versions to detect regressions or inconsistencies [2, 11, 12, 21]. Early work by McKeeman et al.[19] and Zeller et al.[34] highlighted its role in quality assurance. Modern advancements have integrated differential testing into release engineering pipelines for production-like evaluations [1, 2, 20], though these processes still rely on manual interventions, leading to inefficiencies and errors [16].

Recent developments have sought to automate several stages of release engineering by leveraging Artificial Intelligence (AI) and Machine Learning (ML) to streamline and enhance testing processes. These advancements have led to automated tools that prioritize test cases, classify defects, and even predict potential failures in the software [5, 8, 9, 14, 18, 21, 23, 25, 28]. Despite these successes in backend optimizations, much of the focus has been on the functionality of the tools rather than the user experience of the developers who use them [29, 30]. As these tools evolve, there is growing recognition of the importance of integrating Human-Computer Interaction (HCI) principles to enhance usability and developer productivity [1, 8].

AI-driven advancements in differential testing have automated decision-making, but developer-tool interfaces remain static. Tools often lack feedback, adaptive support, and seamless integration into workflows, limiting their utility during high-stakes debugging. Research highlights the need for systems that combine automation with dynamic, intuitive, and interactive support [1, 3, 8, 30]. This has fueled demand for HCI innovations emphasizing contextual awareness, real-time feedback, and adaptive interfaces to meet developers' evolving needs.

In this context, research is beginning to focus on building systems that integrate both AI and HCI principles, thus allowing for more adaptive, context-aware interfaces that support developers at every stage of their work. By merging the power of AI in automating tedious backend processes with HCI's emphasis on developer-centered design, it is possible to create systems that not only improve the efficiency and accuracy of differential testing but also ensure that these tools are usable and effective for developers. This holistic approach is critical in advancing the field, as it addresses not only the technical challenges of differential testing but also the human challenges, ultimately paving the way for more productive and effective software engineering practices.

6 Conclusion

6.1 Limitations

Because the OCEs' work is critical, time-sensitive, and laborious, conducting a study with this population is challenging. While the five participants provided valuable insights, their views and experiences may not represent the entire team, and their voluntary participation is subject to the typical selection bias. We conducted our study collaboratively with a single team, chosen because of its critical mission, its existing investment in differential testing, and its enthusiasm for improving work practices. Our findings may not generalize to other teams doing differential testing, particularly at other companies. Finally, while our prototype used up-to-date LLM models, fine-tuned on recent Diffs from the team's databases, our early-stage prototype does not have the same usability or engineering quality of their established tool, which could have influenced participant feedback.

6.2 Future Work

Building on these findings, future work should focus on enhancing real-time collaboration features such as direct Diff delegation, team-based clustering, and shared annotations. Additionally, incorporating detailed metadata and rationales for AI predictions would address transparency concerns and foster greater trust. Advanced filtering options based on team ownership or historical patterns could further tailor the tool to individual workflows. Expanding the tool's integration with other platforms, such as issue trackers and code repositories, would streamline transitions from debugging to resolution. Conducting longitudinal studies to measure the impact of the redesigned DiffViewer on productivity, developer satisfaction, and software quality over extended periods would provide valuable insights.

6.3 Conclusions

This study presents the potential of combining AI and HCI principles in the design of developer tools. By addressing core pain points such as cognitive overload, inefficiencies in manual labeling, and fragmented workflows, the redesigned DiffViewer represents a significant step forward in differential testing tooling. Features like clustering, AI-based predictions, and an intuitive block-based UI empower software engineers to make faster, more informed decisions. However, balancing automation with transparency and trust remains essential. By addressing current limitations and exploring further enhancements, future iterations of the DiffViewer can serve as a model for intelligent, developer-friendly tools that bridge the gap between technical automation and human usability, advancing release engineering and software development.

References

[1] B. Adams, S. Bellomo, C. Bird, T. Marshall-Keim, F. Khomh, and K. Moir. The practice and future of release engineering: A roundtable with three release engineers. *IEEE Software*, 32(2):42–49, 2015.

[2] B. Adams and S. McIntosh. Modern release engineering in a nutshell – why researchers should care. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 5, pages 78–90, 2016.

[3] C. Amrit and A. K. Narayanappa. An analysis of the challenges in the adoption of mlops. *Journal of Innovation and; Knowledge*, 10(1):100637, Jan. 2025.

[4] S. Asthana, H. Sajnani, E. Voyloshnikova, B. Acharya, and K. Herzig. A case study of developer bots: Motivations, perceptions, and challenges. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2023, page 1268–1280, New York, NY, USA, 2023. Association for Computing Machinery.

[5] M. Bagherzadeh, N. Kahani, and L. Briand. Reinforcement learning for test case prioritization. *IEEE Transactions on Software Engineering*, 48(8):2836–2856, 2022.

[6] T. Besker, A. Martini, and J. Bosch. Technical debt cripples software developer productivity: a longitudinal study on developers' daily software development work. In *Proceedings of the 2018 International Conference on Technical Debt*, TechDebt '18, page 105–114, New York, NY, USA, 2018. Association for Computing Machinery.

[7] A. Carvalho, W. Luz, D. Marcílio, R. Bonifácio, G. Pinto, and E. Dias Canedo. C-3pr: A bot for fixing static analysis violations via pull requests. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 161–171, 2020.

[8] S. Chittala. Aiops and devops: Catalysts of digital transformation in the age of automated operations. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 10(6):155–166, Nov. 2024.

[9] A. Dakkak, J. Bosch, and H. H. Olsson. The role of post-release software traceability in release engineering: A software-intensive embedded systems case study from the telecommunications domain. In *2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 169–176, 2022.

[10] F. M. A. Erich, C. Amrit, and M. Daneva. A qualitative study of devops usage in practice. *Journal of Software: Evolution and Process*, 29(6), June 2017.

[11] R. B. Evans and A. Savoia. Differential testing: a new approach to change detection. In *The 6th Joint Meeting on European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering: Companion Papers*, ESEC-FSE companion '07, page 549–552, New York, NY, USA, 2007. Association for Computing Machinery.

[12] R. B. Evans and A. Savoia. Differential testing: A new approach to change detection. *Software Testing, Verification and Reliability*, 6:549–552, 2007.

[13] F. Fagerholm and J. Münch. Developer experience: Concept and definition. In *2012 International Conference on Software and System Process (ICSSP)*, pages 73–77, 2012.

[14] M. Hausi A., M. Litoiu, L. F. Rivera, M. Rasolroveicy, N. M. Villegas, G. Tamura, I. Watts, E. Erpenbach, and L. Shwartz. Proactive continuous operations using large language models (llms) and aiops. In *Proceedings of the 33rd Annual International Conference on Computer Science and Software Engineering*, CASCON '23, page 198–199, USA, 2023. IBM Corp.

[15] H. Hutchinson, W. Mackay, B. Westerlund, B. B. Bederson, A. Druin, C. Plaisant, M. Beaudouin-Lafon, S. Conversy, H. Evans, H. Hansen, N. Roussel, and B. Eiderbäck. Technology probes: inspiring design for and with families. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '03, page 17–24, New York, NY, USA, 2003. Association for Computing Machinery.

[16] D. Lehmann and M. Pradel. Feedback-directed differential testing of interactive debuggers. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, page 610–620, New York, NY, USA, 2018. Association for Computing Machinery.

[17] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles. A survey of devops concepts and challenges. *ACM Comput. Surv.*, 52(6), Nov. 2019.

[18] T. Mboweni, T. Masombuka, and C. Dongmo. A systematic review of machine learning devops. In *2022 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, pages 1–6, 2022.

[19] W. M. McKeeman. Differential testing for software. *Digital Technical Journal*, 10(1):100–107, 1998.

[20] Y. Noller, C. S. Păsăreanu, M. Böhme, Y. Sun, H. L. Nguyen, and L. Grunske. Hydiff: hybrid differential software analysis. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, page 1273–1285, New York, NY, USA, 2020. Association for Computing Machinery.

[21] M. Openja, B. Adams, and F. Khomh. Analysis of modern release engineering topics : – a large-scale study using stackoverflow –. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 104–114, 2020.

[22] P. Perera, R. Silva, and I. Perera. Improve software quality through practicing devops. In *2017 Seventeenth International Conference on Advances in ICT for Emerging Regions (ICTer)*, pages 1–6, 2017.

[23] A. Podgurski, D. Leon, P. Francis, W. Masri, M. Minch, J. Sun, and B. Wang. Automated support for classifying software failure reports. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, page 465–475, USA, 2003. IEEE Computer Society.

[24] A. Razzaq, J. Buckley, Q. Lai, T. Yu, and G. Botterweck. A systematic literature review on the influence of enhanced developer experience on developers' productivity: Factors, practices, and recommendations. *ACM Comput. Surv.*, 57(1), Oct. 2024.

[25] G. Rothermel, R. Untch, C. Chu, and M. Harrold. Test case prioritization: an empirical study. In *Proceedings IEEE International Conference on Software Maintenance - 1999 (ICSM'99). 'Software Maintenance for Business Change' (Cat. No.99CB36360)*, pages 179–188, 1999.

[26] M.-A. Storey and A. Zagalsky. Disrupting developer productivity one bot at a time. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, page 928–931, New York, NY, USA, 2016. Association for Computing Machinery.

[27] C. Stylianou and A. S. Andreou. Investigating the impact of developer productivity, task interdependence type and communication overhead in a multi-objective optimization approach for software project planning. *Advances in Engineering Software*, 98:79–96, Aug. 2016.

[28] V. M. Tamanampudi. Natural language processing in devops documentation: Streamlining automation and knowledge management in enterprise systems. *Journal of AI-Assisted Scientific Discovery*, 1(1):146–185, June 2021.

[29] M. H. Tanzil, M. Sarker, G. Uddin, and A. Iqbal. A mixed method study of devops challenges. *Information and Software Technology*, 161:107244, Sept. 2023.

[30] S. Tatineni and K. Allam. Ai-driven continuous feedback mechanisms in devops for proactive performance optimization and user experience enhancement in software development. *Journal of AI in Healthcare and Medicine*, 4(1):114–151, Mar. 2024.

[31] A. Tornhill and M. Borg. Code red: the business impact of code quality - a quantitative study of 39 proprietary production codebases. In *Proceedings of the International Conference on Technical Debt*, TechDebt '22, page 11–20, New York, NY, USA, 2022. Association for Computing Machinery.

[32] A. K. Vajjala, A. K. Vajjala, C. Badea, C. Bird, R. DeLine, J. Entenmann, N. Forsgren, A. Hramadski, S. Sanyal, O. Surmachev, T. Zimmermann, H. Mohammad, J. D'Souza, and M. Demyanyuk. Enhancing differential testing: Llm-powered automation in release engineering. In *Proceedings of the IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice (ICSE SEIP)*, page to appear, Ottawa, Ontario, Canada, April 27–May 3 2025. to appear.

[33] W. Wong, J. Horgan, S. London, and H. Agrawal. A study of effective regression testing in practice. In *Proceedings The Eighth International Symposium on Software Reliability Engineering*, pages 264–274, 1997.

[34] A. Zeller and R. Hildebrandt. Simplifying and isolating failure-inducing input. In *Proceedings of the International Conference on Software Engineering*, pages 183–200, 2002.