Linking Software Changes to Incident Reports

Investigating Correlations Between Root Causes and the Mean Time To Repair of Incidents

Danny Bunschoten[1]

Supervisor(s): Prof.Dr.Ir. Diomidis Spinellis[1]**, Eileen Kapel**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Danny Bunschoten
Final project course: CSE3000 Research Project
Thesis committee: Diomidis Spinellis, Eileen Kapel, Benedikt Ahrens

Abstract

The availability and reliability of online systems form the cornerstone of modern civilization. Companies actively try to minimize downtime during incidents, and publishing incident reports afterwards is a standard practice. However, what is missing is an overview of the distribution of the categories of causes leading up to the incident and their characteristics. This paper fills that research gap by answering the question of a relation between different categories of software changes causing the incidents, and their respective mean time to repair (MTTR). A taxonomy for classifying the root causes and time to detect (TTD) of incident reports was derived. A total of 258 publicly available incident reports authored by Google were scraped, and a zero-shot classification model was chosen to classify these. Additionally, the analysis focused on the time to repair (TTR) for each category. This found that incidents caused by software version incompatibilities have the highest MTTR of 69.8 hours, followed by code defects of 54.0 hours, while the other categories have values between 13 and 20 hours. Given that the TTR of an incident is primarily impacted by the number of skilled engineers available, having an estimate of the difficulty based on empirical data could help improve resource distribution based on early indications of root causes.

1 Introduction

Online systems and their continuous availability are critical for performing modern business operations, and a growing number of companies depend on system availability for their core functions [1]. However, incidents are common and tend to negatively affect user experience and consumer trust, resulting in substantial financial losses [2]. For instance, Amazon's downtime of one hour during Prime Day of 2018 is estimated to have cost between $72 million and $99 million [3].

Previous research has explored various approaches to incident management and prediction. Research performed by Google has shown that changes to a live system account for roughly 70% of its incidents [4]. Given this, investigating the various causes of incidents can provide valuable insights into their underlying mechanisms.

Existing literature has shown that using key performance indicators (KPIs) to predict the state of the system provides promising results [5]. However, such approaches are limited by nature due to their constrained inputs; They do not take the changes to the system into account and purely rely on the KPIs. Other research has focused on analyzing the causes of bugs [6, 7], but this is not directly related to incidents in systems. Additional studies have analyzed factors that contribute to the mean time to repair (MTTR) of incidents [8], and concluded that the MTTR is mostly affected by the availability of engineers and their skills.

Despite these contributions, there remains a significant gap in understanding how different categories of software changes correlate with incident characteristics, particularly resolution times. No comprehensive empirical study has systematically categorized software changes that lead to incidents and analyzed their relationship with MTTR in large-scale production environments.

To address this research gap, this paper presents an empirical study to investigate the relationship between software changes and the time to repair of incidents. This study performs a quantitative analysis over a subset of 258 out of a total dataset of 979 incident reports. Every analyzed incident report is categorized by its root cause, and its time to detect (TTD).

This research addresses the following primary research question:

> **To what extent do different types of software changes correlate with the mean time to repair (MTTR) of incidents at Google?**

The following subquestions have been formulated:

> **RQ1:** What is the distribution of software changes categories associated with incidents in the dataset?
> **RQ2:** What is the distribution of time to detect (TTD) across different incident categories?
> **RQ3:** What is the distribution of time to repair (TTR) across different incident categories?

The main contributions of this work include (1) a systematic categorization of software changes leading to incidents in a large-scale production environment, (2) an empirical analysis of the relationship between change categories and resolution times, and (3) insights into the prevalence of latent issues across different change types.

The remainder of this paper is organized as follows: Section 2 reviews previous research on incident analysis and management. Section 3 outlines the methodology used for data collection and labeling. Section 4 discusses ethical considerations and responsible research practices. Section 5 presents the findings from the analysis. Section 6 discusses the implications and limitations of the results. Finally, Section 7 concludes with key insights and suggestions for future research.

2 Related Literature

It is common practice in the software engineering community to share incident reports after successfully resolving an incident, as it allows for the identification of patterns and trends in incidents [9].

Prior research has focused on categorizing the causes of incidents. Researchers have demonstrated that a predictive model can be developed to identify changes that may lead to an incident based on business KPIs [10]. This work laid the foundation for coupling bad changes to their incidents.

Building on this, subsequent work focused on predicting whether a live change to a system would lead to an incident or not [5]. A model called *SCWarn* was proposed that uses a combination of business KPIs, machine KPIs, and logs to predict change-induced incidents. Four categories of change-induced incidents were identified: *configuration*, *code*, *data*, and *infrastructure*.

This taxonomy forms the basis of the current research. However, it is limited to incidents caused by deliberate system changes. Other causes of incidents, such as surges in usage, were not taken into account in the study, nor were they part of the classification scheme. For this reason, a more comprehensive taxonomy is used in this research, which includes additional incident causes.

Moreover, it has been shown that many severe incidents are caused by small changes that have a low TTD [11]. In contrast, *gray failures* represent a class of issues that are not quickly noticeable, e.g. memory leaks, and are paired with a higher TTD. These issues have been shown to be more challenging to detect using conventional automatic failure detection systems [12].

Human factors also contribute to the outcome of an incident. Less experienced developers or those unfamiliar with the codebase are more likely to introduce bugs [7]. The number of available engineers and their respective skills are the main contributing factors to the MTTR of an incident [8].

Although some valuable insight can be gained from these studies, they are limited by only taking into account incidents that are caused by deliberate changes. Incidents caused by latent issues or external factors are excluded from their analyses. This research aims to broaden the analysis by including all incidents, regardless of their cause, and correlating this to the TTD and TTR of the incidents. Knowing that the number of available skilled engineers is a main contributing factor, this knowledge can help the industry allocate resources more efficiently during incident response, aimed at lowering the MTTR.

3 Methodology

This research investigates whether there is a correlation between the MTTR and the cause of incidents reported by Google. To answer this question, a publicly available dataset of incident reports from Google's service status platform was collected and analyzed. Google's data on incident resolution was chosen due to its diversity and transparency: the dataset contains 979 incidents across 191 unique services and provides a clear TTR per incident. Out of this initial dataset, a selection of 258 reports was made for further analysis. A total of 50 reports were manually categorized based on the root cause that led to the incident and the TTD. The dataset was then used to train a classifier to categorize the remaining incident reports into these categories automatically.

3.1 Data Collection

For this research, Google's publicly accessible data on their incident reports was utilized [13]. While Google does provide a structured format of data about their incident reports [14], this dataset is limited to only the most recent 132 reports as of May 5, 2025. To access the full dataset of all available incident reports, a custom web scraping approach was employed. The code used for this scraping has been publicly posted [15]. On its website, Google has divided its incident reports into 191 categories, where every category corresponds to one of its products. The script initially visited the overview page and then collected the links for every individual category. Then, it systematically went through every individual category and collected the URL of all incident reports related to that category. Finally, the raw HTML content of every incident report was downloaded and saved using its unique identifier (ID). This was done to prevent storing duplicate incident reports in case an incident report would belong to multiple categories. Eventually, 979 reports were downloaded on 6 May 2025.

3.2 Data Preprocessing

Preprocessing was necessary to extract relevant information, reduce document size, and ensure compatibility with downstream models. This process occurred in several stages. First, metadata extraction was performed for each incident report. Since the incident reports followed a consistent HTML structure, the title, start time, and end time were extracted deterministically. Non-informative elements such as `<script>` and `<style>` tags, and boilerplate content were removed to minimize noise in the dataset.

A qualitative review identified that the term *root cause* was consistently used in reports that provide causal information. Consequently, a case-insensitive regular expression was employed to identify these sections, accounting for variations such as hyphenation. This filtering criterion reduced the dataset from 979 to 258 reports containing explicit root cause descriptions.

Within these filtered reports, the content was further narrowed to focus solely on sentences surrounding the incident's root cause mention, as preliminary examination revealed these segments contained the most classification-relevant information. Given that the eventual model used had a limit of 1024 characters, this noise reduction was essential.

3.3 Data Labeling

Incidents were categorized along two dimensions: (1) six categories based on the type of software change leading to the incident, and (2) three categories based on whether the issue was latent. The TTRs of these incidents were extracted, and the distribution was analyzed for each category. This approach enabled the identification of issues and allowed the industry to more effectively distribute its resources when addressing these problems.

Validated categories for categorizing incident reports were adopted and slightly adapted [5]. Most notably, the previous research focused on change-induced incidents; the taxonomy used in their research only entails categories for change-induced incidents, e.g. there is no category for incidents caused by a power outage. Adaptations were made to include proper categories for non-change-induced incidents. In the end, the following taxonomy was developed:

- **Code Defect**: A logical or implementation error in the source code.
- **Configuration Error**: A faulty or unintended change in system configuration.
- **Resource Contention**: A system failure due solely to excessive load or traffic, excluding cases where high load merely revealed an underlying bug.
- **Dependency Version**: A compatibility issue caused by updates to third-party packages or dependent systems.

- **No Software**: The root cause lies outside of the software stack (e.g., hardware faults, networking failures).
- **Unclear**: Reports that mention a "root cause" but do not provide a specific or classifiable explanation.

For categorizing the TTD of the issue, three labels were used. Notably, labeling this required relying on the semantic content of the report rather than explicit information about the timing of the root cause related to the incident. For example, statements such as *due to a recent change*, or *was quickly rolled back* were used to deduce the categories. Because of this, the following inexact labeling scheme was chosen.

- **Low TTD**: The incident was quickly detected after the root cause had happened (e.g., a rollout that contained a bug).
- **High TTD**: The incident occurred significantly later than the root cause (e.g., a race condition that was not immediately apparent).
- **Unknown**: The timing of the root cause was not stated in the report.

3.4 Automated Labeling via a Custom Classifier

To estimate the performance of the model, a comparison to 50 manually labeled incident reports was made. After downloading all incident reports, the unique IDs of the reports were sorted alphabetically, such that the order in which they were downloaded or listed did not impact the outcome. Subsequently, the keys were shuffled using Python's `random.shuffle` function, with seed *42*. Following this, the top 50 incident reports were selected and manually annotated.

After manually labeling 50 incident reports, several classification algorithms were experimented with to yield the best performance. Initially, a simple **bag-of-words (BoW)** strategy with both **support vector machines (SVM)** and **logistic regression (LR)** was used for both classification tasks. Similarly, the **term frequency–inverse document frequency (TF–IDF)** model was used on both classification tasks. Furthermore, to ensure a comprehensive analysis, more complex supervised **transformer**-based classification models [16] were evaluated, beginning with **BERT** [17]. Subsequently, smaller transformer models, such as **RoBERTa** [18] and **DistilBERT** [19], were tested. Finally, **zero-shot classification** [20]—a deterministic, reproducible model capable of performing classifications without requiring large training sets—was experimented with.

After implementing the abovementioned algorithms, an evaluation was made on three evaluation metrics. As shown in Equations (1)–(3), the classifiers were evaluated using accuracy, precision, and Cohen's kappa (κ). The observed agreement p_o and expected agreement p_e, defined in Equations 4 and 5, were used to compute κ. Unlike accuracy and precision, Cohen's kappa compensated for agreements between two ratings that might occur by chance. This gave a more thorough indication of the performance of the model. Eventually, the best-performing model was chosen and used for the rest of the evaluation.

$$\text{accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \tag{1}$$

$$\text{precision} = \frac{TP}{TP+FP} \tag{2}$$

$$\kappa = \frac{p_o - p_e}{1-p_e} \tag{3}$$

$$p_o = \frac{TP+TN}{TP+TN+FP+FN} \tag{4}$$

$$p_e = \frac{(TP+FP)(TP+FN)+(TN+FN)(TN+FP)}{(TP+TN+FP+FN)^2} \tag{5}$$

The zero-shot model used, *facebook/bart-large-mnli*[1], allowed for tuning via *hypothesis template* and *labels*. In this setup, the model laid the classification out as a natural language inference (NLI) problem. For every text the model tried to classify, the label was placed in the hypothesis template, forming a coherent sentence (e.g. "The issue explained in the incident report was explicitly mentioned to have been caused by a *label*."). The model then calculated the probability for *entailment* and *contradiction*, and these were converted to probabilities per label. Following this, an argmax was used to pick the label with the highest probability.

Given the model's dependence on the combination of hypothesis templates and labels, several setups were tried and an analysis was conducted to compare their performance using the predefined evaluation criteria. For category prediction, the following hypothesis template was employed: *According to the given text, the explicitly stated root cause of the failure is:* {}. Similarly, for the TTD prediction, the used hypothesis template was: *The incident was caused by:* {}. The labels that have been used for the research are shown in Table 1 and Table 2.

4 Responsible Research

Performing responsible and reproducible research is crucial for ensuring the paper benefits the broader research community. This section outlines the ethical considerations taken during this research.

4.1 Ethical Considerations

For this research, Google's publicly available dataset on incident reports was used. Due to a lack of an API for retrieving the required data, a custom web-scraping technique was utilized. To ensure the website was not overloaded during the scraping process, rate limiting was used, with an interval of 10 seconds between every request. Furthermore, extra care was taken to inspect the `robots.txt` file before scraping [21], which indicated no issues with the planned scraping. Given that the scraped data did not contain any personal identifiable information (PII), and did not form any privacy-related issues, no anonymization procedures were required. While scraping the website, no restrictions were imposed.

[1] https://huggingface.co/facebook/bart-large-mnli

Table 1: Categories and their label used in model predictions.

Category	Label Used
code defect	a concrete code bug or flaw in source code causing system malfunction, could also be a rollout
configuration error	misconfigured system settings or incorrect parameters
resource contention	sudden overload or increase in traffic possibly leading to a resource contention like CPU, memory, or disk
no software	an issue outside software e.g. hardware failure or power loss
unclear	insufficient information to categorize the root cause, or not matching any known categories
software version	software or library version incompatibility

Table 2: TTD categories and labels used in model predictions.

Category	Label Used
low TTD	recent change or event
high TTD	existing issue
unknown	something unrelated to a change or unknown

4.2 Reproducibility

To promote reproducibility and transparency, the code used for the retrieval of the data has been open-sourced under the MIT license [15]. This decision was made to ensure that other researchers can replicate the results of this study and build upon it while keeping the data with its original distributor. However, detailed instructions are available to enable other researchers to reproduce the same scraping environment used in the original research.

4.3 Biases

Given that the dataset is publicly available, researchers must be cautious of potential selection biases in reporting, e.g. Google might have chosen not to disclose embarrassing reports. Furthermore, since there are 191 unique categories of services reporting their incidents, the distribution might be biased by different reporting rates across divisions.

4.4 The use of AI

AI tools have been used to improve parts of this paper. Specifically, Claude (Sonnet 4 and Opus 4) has been used to fix LaTeX errors that arose during the writing of this paper. Claude has also been used for refining certain figures presented in this paper. ChatGPT and Grammarly were utilized for spell-checking and enhancing the overall readibility of this paper. A list of prompts used during the development of this paper is provided in Appendix A.

5 Results

The incident reports analyzed in this study have been partially manually annotated; the remaining reports have been labeled using a custom classifier. This section presents the results of various classifiers that have been tested, along with statistical insights derived from the labeled incident reports.

5.1 Automatic Labeling

After manually labeling 50 incident reports, several machine learning algorithms were experimented with and evaluated on three evaluation metrics. The results of these algorithms are shown in Table 3 and Table 4. The zero-shot classification algorithm showed the best performance across all three evaluation metrics. Hence, it was decided to continue with zero-shot classification to classify the remaining 208 incident reports. In total 258 labeled incident reports were used for further analysis, of which 50 were manually annotated, and 208 automatically labeled.

Table 3: Evaluation metrics of machine learning algorithms on category prediction.

Algorithm	Accuracy	Precision	Cohen's Kappa
BoW (LR)	0.47	0.49	0.35
BoW (SVM)	0.47	0.41	0.35
TF-IDF (LR)	0.18	0.03	0.0
TF-IDF (SVM)	0.18	0.03	0.0
BERT	0.12	0.02	-0.1
RoBERTa	0.29	0.09	0.0
DistilBERT	0.29	0.09	0.0
Zero-shot classification	0.88	0.89	0.85

Table 4: Evaluation metrics of machine learning algorithms on TTD prediction.

Algorithm	Accuracy	Precision	Cohen's Kappa
BoW (LR)	0.76	0.76	0.26
BoW (SVM)	0.71	0.75	0.18
TF-IDF (LR)	0.71	0.50	0.0
TF-IDF (SVM)	0.71	0.50	0.0
BERT	0.71	0.50	0.0
RoBERTa	0.71	0.49	0.0
DistilBERT	0.71	0.49	0.0
Zero-shot classification	0.89	0.92	0.71

5.2 Category Distribution

Having labeled the incident reports, an analysis on the distribution of categories of incidents at Google has been created.

Among the 258 incident reports, the three most frequent categories were *resource contention*, *code defect*, and *configuration error*. This aligns with findings from earlier research on change-induced incidents [22], which found code defects and configuration errors to be the most common causes. Additionally, 56 reports (21.7%) could not be classified, and were labeled as *unclear*. The full distribution is shown in Figure 1.

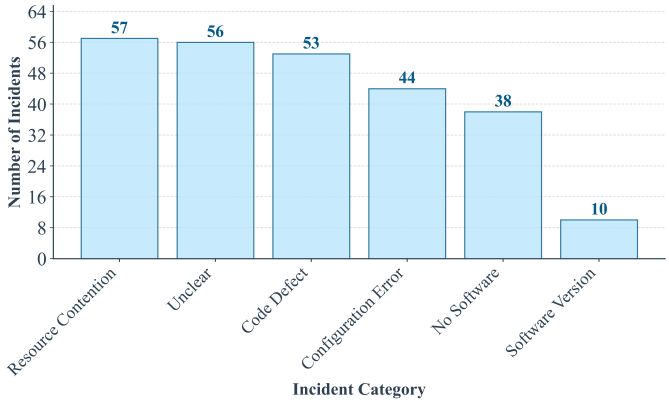

Figure 1: Distribution of categories of incidents at Google.

5.3 Time To Detect per Category

An analysis of the TTDs of the issues reveals that some categories of issues are earlier recognizable than others. For example, incidents belonging to the *no software* category had a low TTD in 20 of the 38 studied cases (52.6%). In contrast, 43 of the 53 *code defect* incidents (81.1%) and 33 of the 44 *configuration error* incidents (75.0%) were categorized as low TTD. Figure 2 illustrates the distribution of latencies of issues that formed the root cause of incidents at Google.

Further examination of the data indicates that incidents with a lower TTD have a higher MTTR (27.8 hours) than their counterparts with a higher TTD (24.5 hours). Furthermore, as shown in Figure 3, the MTTR of the high TTD incidents is impacted by a severe outlier of 1063.2 hours. To assess the impact of extreme values on the MTTR, both categories had their single highest outlier removed, which is common practice in studies on incident time [23]. This adjustment reduced the MTTRs to 25.1 hours for the lower and 11.7 hours for the higher TTD.

As shown in Figure 4, the lower TTD category shows a larger distribution of TTR than the higher TTD category.

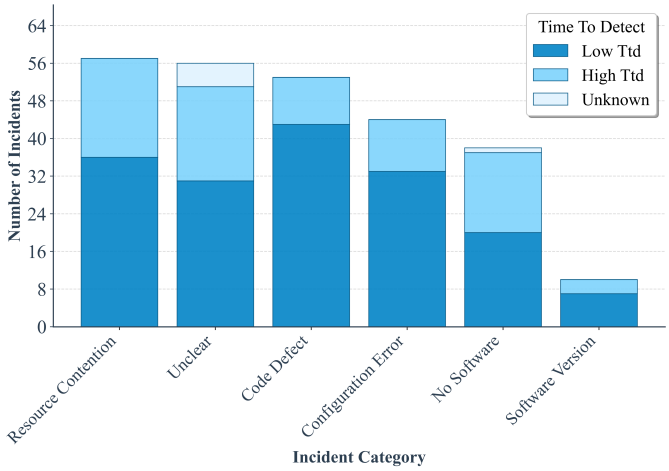

Figure 2: Distribution of latencies of issues that formed the root cause of incidents at Google.

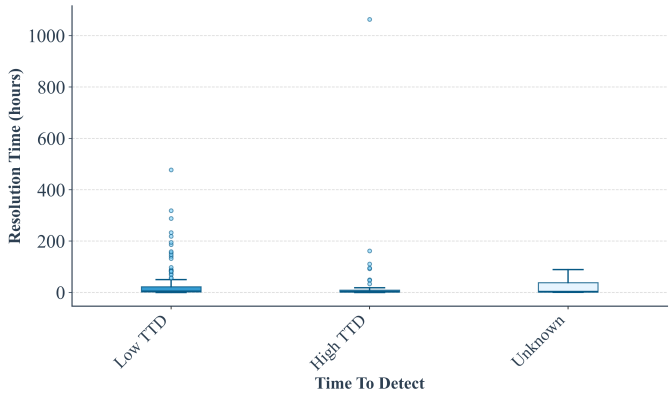

Figure 3: TTR per TTD Category.

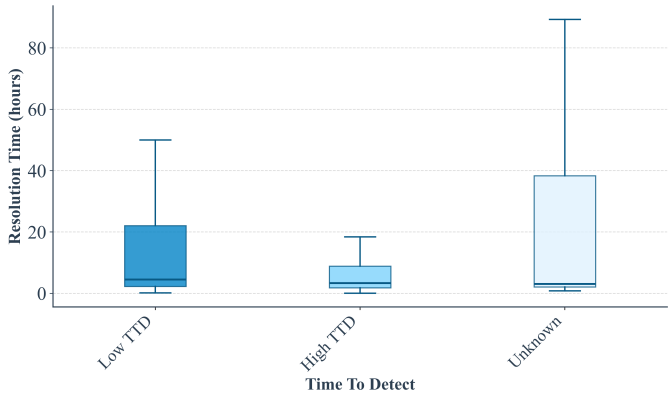

Figure 4: TTR per TTD category, without any outliers drawn.

5.4 Distribution of Time To Repair per Category

When comparing the MTTR of incidents, the *software version* and *code defect* categories had the highest MTTR.

Notably, as shown in Table 5, the median times to repair of the categories range from 3 to 6 hours. All categories show a significantly higher mean than median, and the *software version* and *code defect* categories show a mean of 69.8 and 54.0 hours respectively. This shows that the data is skewed to the right, indicating that there are a few incidents with a high TTR.

Table 5: MTTR statistics by incident category

Category	Mean (hrs)	Median (hrs)
Code Defect	54.0	3.5
Configuration Error	21.9	4.1
No Software	13.5	5.8
Resource Contention	16.0	4.1
Software Version	69.8	3.4
Unclear	16.9	3.7

As shown in Figure 5, extreme outliers dominate the data and skew the MTTR of categories to the right. Similarly to

the TTD analysis, the single-most extreme value was left out of every category to assess the effect of outliers. This reduced the *software version* category from 69.8 to 51.6 hours and the *code defect* category from 54.0 hours to 34.6 hours. While this reduction does lower the MTTR of both categories, their adjusted MTTR still remains significantly higher than the remaining categories. This shows that while outliers do impact the data, there is no single outlier accountable for this skew right.

Figure 6 shows the distribution without outliers. This shows that the software version category follows another distribution than the other five categories.

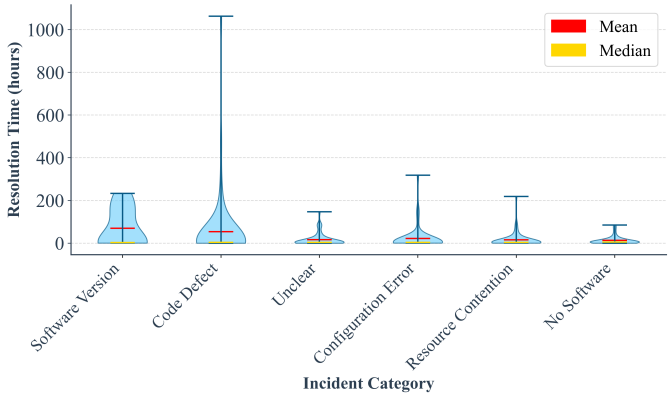

Figure 5: Violin plot of TTR of incidents at Google.

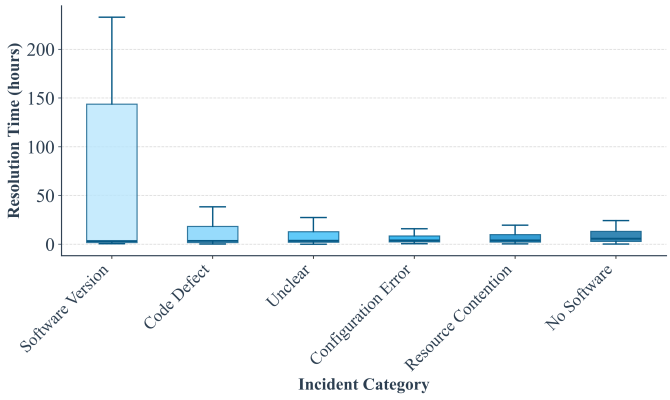

Figure 6: Box plot of TTR of incidents at Google without outliers.

6 Discussion

This section outlines how these findings relate to other research. Finally, this section discusses the potential limitations of this study.

6.1 Distribution of Causes

The distribution of causes leading to system failure, as shown in Figure 1, differs from the distribution reported by Zhao et al. [5]. In particular, we found the most common cause of system failure to be *resource contention*, accounting for 57 of the 258 analyzed failures (22.1%). This differs from the results found by Zhao et al., where the most common cause is *code defect*, accounting for 38% of the analyzed cases. Furthermore, the second most common cause in their research is *configuration error*, accounting for 31% of the analyzed cases. This discrepancy likely arises because Zhao et al. only examined incidents caused by deliberate changes to the system when conducting this analysis. This is likely to explain the differences between results, given that frequent cases of sudden high load causing issues are excluded in their research, while these are included in our research.

In the distribution of categories leading to incidents, we found that software version mismatches are very uncommon compared to other categories. This observation may be explained by the nature of how such mismatches arise, typically as a result of dependency upgrades [24]. These dependencies can vary widely, ranging from code libraries to broader components such as frameworks and operating systems. However, in large, mature production environments, where high availability is crucial, code changes are more common than dependency upgrades. As a result, the opportunity for version mismatches to occur is inherently lower, which may account for their lower frequency.

6.2 Time To Detect by Category

Previous research by Wu et al. has found that change-induced incidents take longer to discover than non-change-induced incidents [22]. The reason given for this is that changes to the system also tend to introduce new metrics that require monitoring. This monitoring can take time and lead to a longer TTD.

When comparing the results found in our research to those found by Wu et al., a translation step is required. Since our research utilizes textual, publicly available incident report data, which lacks exact information on the TTD, linguistic hints are used to categorize based on the TTD. In our research, a discrete scale is used to express the TTD, Because of this, no exact measures exist on what is considered as *low TTD* versus *high TTD*. Another difference between our research and the existing research is the form of categorization. Whereas the research performed by Wu et al. uses two categories: change-induced and non-change-induced. Our research employs six categories, where *code defect*, *configuration change* and *dependency upgrade* tend to be considered as more change-induced, and *resource contention* and *no software* could be considered as less change-induced. When translating as described, the change-induced group has 77.6% of change-induced incidents belonging to the category *low TTD*, whereas only 58.3% of the non-change-induced incidents belong to the *low TTD* category. Research by Wu et al. found the 75th percentile of TTD for change-induced incidents to be 26.8X of that of non-change-induced incidents. This suggests that the found timings of the detectability align with those found by Wu et al. research.

Furthermore, the *configuration change* and *code defect* categories exhibit a high percentage of incidents with low TTD, 81.1% and 75.0% respectively. This aligns with earlier findings [11, 25], which found that a large number of change-induced incidents are caused by small changes with a low

TTD. A common recommendation to detect defects, is to make use of CI/CD pipelines [26] and canary roll-out strategies [27]. These strategies help detect flaws by reducing the exposure to customers, by either testing before deployment, or by slowly rolling changes out to customers. These approaches are especially useful for detecting defects with a low TTD, as is the case with *configuration changes* and *code defects*.

6.3 Resolution Time By Category

As shown in Figure 4, the distribution of the TTR for the *software version* category significantly differs from the other categories. The *software version* category exhibits an interquartile range of 142.2 hours, which is substantially wider than all other categories. This can be explained by the nature of this category, *software version* mismatches are expected to be more complex and time-consuming to resolve [28]. However, the small number of datapoints (n=10) in this category does influence the confidence of this finding, since future work is required to further validate this finding.

6.4 Limitations

The limitations and threats to validity are classified into *internal validity*, *external validity*, and *reliability* [29].

- **reliability**: All manual labeling was done by one single researcher. This introduces the risk of a subjective bias. Future work could improve upon this by using multiple annotators and cross-validation techniques to ensure the reliability of the results.
- **internal validity**: All data used was publicly available data. Given that the incident reports come from 191 unique products, where, given the size of Google, all products are likely to be supervised by their own manager. It is likely that there exists variability in the reporting rate, or depth of reporting across products.
- **internal validity**: In some cases, incidents can be underreported or selectively reported, especially when an incident had an embarassing cause. For example, an incident could merely state that there was downtime, and that a fix has been implemented. Such incomplete disclosures would cause them to be excluded from this research, this introduces a reporting bias.
- **external validity**: All data originates from a single company. While Google has a diverse portfolio of 191 services, they are a large-scale experienced company, proven to be capable of consistently delivering complex services to customers. The results of this study may not be representative for all organizations, in particular smaller companies with less experience. Further research is required to validate these findings.

7 Conclusion

This paper utilizes publicly available incident reports to examine the distribution of incidents at Google, their time to detect (TTD), and their mean time to repair (MTTR). The analysis entailed classifying 258 publicly available incident reports.

While the findings do provide valuable insights that align with results found in previous research, they are limited by the constrained data used in the study. All data used originates from a single company; this may not be representative of all organizations. Future work should try to validate this research by having a more diverse data set.

This analysis revealed that out of the analyzed incidents, *resource contention*, *code defects*, and *configuration errors* were the most common root cause. This aligns with earlier research, which found *code defects* and *configuration errors* as the most common root causes when considering only change-induced incidents [5]. Further analysis of the MTTR revealed that the *software version* category (69.8 hours mean) and the *code defect* category (54.0 hours mean) exhibited the highest MTTR, while the other categories ranged from 22 hours to 13 hours. Recognizing that the *software version* and *code defect* categories have the highest MTTR, large-scale organizations should consider allocating additional skilled engineering resources to these incident types to reduce their resolution times.

Moreover, this research supports earlier findings [11, 25], indicating that incidents are commonly caused by small changes that are quickly detectable. In this research, *code defects* and *configuration errors* were identified as common root causes, where for both categories, at least 75% of the incidents were classified as *low TTD*. Knowing that a significant subset of the analyzed incidents exhibited a low TTD, introducing CI/CD pipelines and canary rollouts is a recommended improvement to reduce the number of customers exposed to faulty rollouts before they can be detected.

References

[1] IBISWorld, "Percentage of business conducted online." https://www.ibisworld.com/us/bed/percentage-of-business-conducted-online/88090/, 8 2024. Accessed: 2025-05-12.

[2] Cloudflare, "How website performance affects conversion rates." Accessed: 2025-05-12.

[3] S. Wolfe, "Amazon's one hour of downtime on prime day may have cost it between $72 million and $99 million in lost sales," *Business Insider*, July 2018. Accessed: 2025-04-30.

[4] B. Beyer, C. Jones, J. Petoff, and N. Murphy, *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly, 2016.

[5] N. Zhao, J. Chen, Z. Yu, H. Wang, J. Li, B. Qiu, H. Xu, W. Zhang, K. Sui, and D. Pei, "Identifying bad software changes via multimodal anomaly detection for online service systems," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2021, (New York, NY, USA), pp. 527–539, Association for Computing Machinery, 2021.

[6] Z. Ni, B. Li, X. Sun, T. Chen, B. Tang, and X. Shi, "Analyzing bug fix for automatic bug cause classification,"

Journal of Systems and Software, vol. 163, p. 110538, 2020.

[7] L. An and F. Khomh, "An empirical study of crash-inducing commits in mozilla firefox," in *Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering*, PROMISE '15, (New York, NY, USA), Association for Computing Machinery, 2015.

[8] A. Aguilar, "Lowering mean time to recovery (mttr) in responding to system downtime or outages: An application of lean six sigma methodology," in *13th Annual International Conference on Industrial Engineering and Operations Management, https://doi.org/10.46254/AN13*, vol. 20230039, 2023.

[9] J. Allspaw, "Blameless postmortems and a just culture," 5 2012.

[10] Y. Zhao, L. Jiang, Y. Tao, S. Zhang, C. Wu, Y. Wu, T. Jia, Y. Li, and Z. Wu, "How to manage change-induced incidents? lessons from the study of incident life cycle," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 264–274, 2023.

[11] Z. Li, Q. Cheng, K. Hsieh, Y. Dang, P. Huang, P. Singh, X. Yang, Q. Lin, Y. Wu, S. Levy, *et al.*, "Gandalf: An intelligent,{End-To-End} analytics service for safe deployment in {Large-Scale} cloud infrastructure," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pp. 389–402, 2020.

[12] P. Huang, C. Guo, L. Zhou, J. R. Lorch, Y. Dang, M. Chintalapati, and R. Yao, "Gray failure: The achilles' heel of cloud-scale systems," in *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, HotOS '17, (New York, NY, USA), p. 150–155, Association for Computing Machinery, 2017.

[13] "Google Cloud Service Health." https://status.cloud.google.com/, 2025. Accessed: 2025-05-05.

[14] "Google Cloud Incidents JSON Feed." https://status.cloud.google.com/incidents.json, 2025. Accessed: 2025-05-05.

[15] D. Bunschoten, "Reproducibility package for incident classification," May 2025. https://doi.org/10.5281/zenodo.15396760.

[16] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-Art Natural Language Processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Association for Computational Linguistics, Oct. 2020.

[17] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[18] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," 2019.

[19] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter," *CoRR*, vol. abs/1910.01108, 2019.

[20] W. Yin, J. Hay, and D. Roth, "Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach," *CoRR*, vol. abs/1909.00161, 2019.

[21] The Web Robots Pages, "robots.txt." https://www.robotstxt.org/, 2025. Accessed: 2025-04-30.

[22] Y. Wu, B. Chai, Y. Li, B. Liu, J. Li, Y. Yang, and W. Jiang, "An empirical study on change-induced incidents of online service systems," in *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 234–245, 2023.

[23] Z. Chen, Y. Kang, L. Li, X. Zhang, H. Zhang, H. Xu, Y. Zhou, L. Yang, J. Sun, Z. Xu, Y. Dang, F. Gao, P. Zhao, B. Qiao, Q. Lin, D. Zhang, and M. R. Lyu, "Towards intelligent incident management: why we need it and how we make it," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, (New York, NY, USA), p. 1487–1497, Association for Computing Machinery, 2020.

[24] T. Dumitraş and P. Narasimhan, "Why do upgrades fail and what can we do about it?," in *Middleware 2009* (J. M. Bacon and B. F. Cooper, eds.), (Berlin, Heidelberg), pp. 349–372, Springer Berlin Heidelberg, 2009.

[25] H. S. Gunawi, M. Hao, R. O. Suminto, A. Laksono, A. D. Satria, J. Adityatama, and K. J. Eliazar, "Why does the cloud stop computing? lessons from hundreds of service outages," in *Proceedings of the Seventh ACM Symposium on Cloud Computing*, SoCC '16, (New York, NY, USA), p. 1–16, Association for Computing Machinery, 2016.

[26] M. Shahin, M. Ali Babar, and L. Zhu, "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices," *IEEE Access*, vol. 5, pp. 3909–3943, 2017.

[27] M. Lindon, C. Sanden, and V. Shirikian, "Rapid regression detection in software deployments through sequential testing," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, (New York, NY, USA), p. 3336–3346, Association for Computing Machinery, 2022.

[28] P. Abate, R. Di Cosmo, G. Gousios, and S. Zacchiroli, "Dependency solving is still hard, but we are getting better at it," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 547–551, Feb 2020.

[29] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, *et al.*, *Experimentation in software engineering*, vol. 236. Springer, 2012.

A Generative AI Prompts

The following prompts were used while the writing the paper:

- Can you see why the references are not rendering properly?
- Should i capitalize ...
- how to get a top 1 next to my name in latex so i can indicate the school using a 1 for all people
- say i have a sentence like this, then i struggle a bit with the plurar which i think is needed here: ...
- Is this sentence gramatically correct? ...
- I dont see what im doing wrong, can you explain? "sections/related.tex", line 15: missing \after "e.g."
- please help me format this nicely: ...
- please help me refenence this ...
- whats wrong with my latex? ...
- can you ensure that my table takes the full width of the page, now it only takes up one column of my page: ...
- please check the text for spelling and grammatical errors: ...
- can you suggest improvements for improving the flow: ...
- can you explain why they use the bottoms array when creating stacked bar charts: ...