Behind the Hot Fix: Demystifying Hot Fixing Industrial Practices at Zühlke and Beyond

Carol Hanna[1], David Elliman[2], Wolfgang Emmerich[1,2], Federica Sarro[1], Justyna Petke[1]
[1]University College London, United Kingdom
[2]Zühlke Engineering Ltd, United Kingdom

Abstract

Rushing a hot fix and having it fail can severely damage a software company's reputation, impacting user satisfaction and future business opportunities. Ensuring best practices for emergency bug handling is critical, yet the process remains elusive in the industry.

We are the first to conduct a study to gain insights on hot fixing industrial practices. We surveyed 24 employees of Zühlke, a mid-sized IT company specializing in providing software engineering services to clients from different domains. We also surveyed 136 software practitioners from a wide range of companies, roles, and geographical areas through an online questionnaire, most having over 10 years of professional experience. Among others, we found that terminology around hot fixing is inconsistent; 56.7% of questionnaire participants stated automated tooling exists for hot fix deployment, but only 25% responded that some tooling is available for hot fix generation; Zühlke employees reported significantly faster hot fix resolution times, attributed to their strong emphasis on agile practices, averaging hours vs. days in other companies.

Based on our study's results we offer key recommendations for both software engineering researchers and industry practitioners.

CCS Concepts

• **Software and its engineering → Maintaining software**.

Keywords

Hot fix, Hot patch

1 Introduction

We have observed the effect that critical software issues can have globally. Just this year, the world came to a halt when a software bug in the CrowdStrike security software affected Microsoft systems [13, 23, 37]. This severe IT outage affected airports, healthcare services, banks, government services, stock markets, and many other sectors [26]. Although this incident gained massive media attention [9], it was not an overly surprising occurrence for industry professionals. As we know, it is common that software bugs end up in production. There is no way to guarantee that we are shipping completely bug-free code, since testing is incomplete, deployment deadlines are tight, and systems are growing larger and larger.

When critical software bugs are discovered in production, generating a fix for them becomes time-sensitive. These bugs can affect important stakeholders, a large number of users, system security, important functionality, etc. The activity of fixing these time-critical bugs is referred to as *hot fixing*, although precise definitions in the literature vary [16]. Here we use the term *hot fix* to mean "an improvement to a specific time-critical issue deployed to a software system in production". Hot fixes aim to mitigate the symptoms of the critical issue at hand as quickly as possible and are not necessarily root-cause fixes. Due to time limitations, these hot fixes might also not be as comprehensively tested as traditional bug fixes. At the same time, their validity is equally, if not more, important to ensure no further reputational damage falls back on the enterprise.

Deploying a hot fix can sometimes backfire, causing more disruption than the original issue. One notable case where a hot fix deployment went wrong involved the video game "No Man's Sky" [2]. Shortly after its release, the developers at Hello Games deployed a hot fix to address several critical bugs and performance issues. However, the hot fix introduced new problems, including game crashes and corrupted save files [3]. This led to significant frustration among players and required additional patches to resolve the new issues. This incident, among many others, stresses the need for careful testing and deployment of hot fixes.

In this paper, we aim to make a stride towards understanding how software professionals handle hot fixing scenarios today. We investigate the current industry standards around this topic, the bottlenecks (if any), and what can be done to optimize hot fixing as a practice in the future. This study is in collaboration with, *Zühlke*, a mid-sized IT company (1K- 5K employees) that provides IT solutions to clients from different sectors. The company is engineering-focused as they specialize in software engineering services to create products from the initial ideation stage to development to deployment for their different clients making them an ideal candidate for investigating industry standards for a software engineering activity. Furthermore, the products they work on span various domains given their diverse clientele, thus making our study not domain-specific and representative of wide views within the software industry. In addition, we distributed the same survey online to gain additional insights from different company profiles. This resulted in 136 more valid responses from software professionals.

Our results shed light on certain aspects of the hot fixing process for which we observed near-consensus such as the average time it takes to hot fix, the number of people involved in hot fixing, and how the need for a hot fix is most commonly discovered. However, the survey also unveils different practices on some matters such as who is responsible for the hot fix, and the description of the hot fix pipeline. We emphasize these differences and strongly encourage the academic community to focus research efforts on them. At the same time, we call on the professional community to collaborate in making these advancements achievable. We hope that this paper facilitates a bridge between academic researchers and industry professionals on the topic to allow for a positive feedback loop and better emergency handling procedures in the future.

In summary, our paper makes the following contributions:

- It is the first research on the current hot fixing practices within the software industry;

- it provides an investigation into a mid-sized software company's current standards on hot fixing;
- it discusses the results of a large-scale questionnaire on hot fixing answered by 136 participants from varied domains, roles, and backgrounds;
- it discusses implications for researchers and software practitioners to help bridge the gap between industry and academia.

2 Research Questions

Industrial hot fixing practices remain elusive in the literature [15]. To explore the current industrial practices for hot fixing and directions for improvement, we conducted a survey to answer the following research questions. These research questions were chosen to provide a comprehensive understanding of the hot fixing landscape with the goal of identifying both challenges and opportunities for improving hot fixing processes in software engineering.

RQ1 [Terminology]: How consistent is hot fixing terminology among software professionals? Consistent terminology is essential to develop productive processes. We aim to investigate how software professionals across different companies, domains, and backgrounds refer to the activity of hot fixing.

RQ2 [Current practice]: What are the current hot fixing practices? This includes understanding what the hot fix process looks like, how much it varies depending on the application domain, how many people are involved and their roles and expertise, and how much the process varies from the usual bug fixing process.

RQ3 [Tooling & Automation]: What is the current level of automation and tooling for hot fixing? The hot fixing process includes many steps such as identifying the need for a hot fix, generating the hot fix, verifying it, and deploying it. We aim to understand the current state of automation and existing tooling for each step in the hot fixing pipeline.

RQ4 [Standardisation]: How standardized are the hot fixing practices across different company and participant profiles? It is not yet clear whether the hot fixing process for software is standardized. We investigate this question from two angles: different company and different participant profiles, to see how they affect the results of our study.

3 Methodology

We present our methodology for understanding the industrial practices for hot fixing and answering our RQs. In particular, we disseminated a questionnaire, which can be found in our artefact [1].

3.1 Questionnaire Design

The questionnaire we designed consists of 38 questions. We include branching such that a participant who is currently working in a software company would be asked about their current company, and a participant who was working but is no longer employed within the software industry would be asked about their most recent role. Thus, a participant would answer a maximum of 28 questions in total. The questionnaire can be answered in around 10 minutes. It is hosted on Microsoft Forms and is anonymous. We organised the questionnaire into four sections: profile questions; hot fix terminology questions; current hot fixing practices questions; and questions about hot fix automation.

In our questionnaire we included a series of questions desinged to collect demographic information from respondents, such as work experience and role within the organization. The inclusion of this information allows us to perform subgroup analyses, uncovering potential differences between various demographic groups.

We implemented branching in Microsoft Forms to ensure that participants who indicated in their first question that they had never worked in a professional software company would automatically end the questionnaire at that point, thus preventing them from accessing and answering the remaining questionnaire questions.

After this first check-point question, we delve into questions around terminology to answer RQ1. At the end of this section, we include an 'attention' question to ensure that the participant is reading the questions and answering coherently (refer to Section 7). After completing the terminology section, participants are presented with the definition of *hot fix* as "an improvement to a specific time-critical issue deployed to a software system in production". This ensures that all respondents consistently understand the terminology when answering subsequent questions. To answer RQ2, the following section focuses on current practices related to hot fixing, including aspects such as frequency, responsible personnel, and the general pipeline. The questionnaire concludes with questions about the tools used in the hot fixing process, specifically inquiring whether certain steps are automated within their organization pertaining to RQ3. Finally, given that we collect demographic information, we can perform a meta-analysis to address RQ4. This comprehensive approach aims to gather insights on both the procedural and technical dimensions of hot fixing.

Before public distribution, we conducted a testing phase to refine the questionnaire. Two authors not involved in the initial design completed it as participants and provided feedback to the original author for improvement. In a second round, we circulated the questionnaire within our research group for further input. All responses from the testing phase were excluded from the study data.

3.2 Questionnaire Dissemination

We distributed the questionnaire to the Zühlke employees via their internal communication channel. Employees were informed that the questionnaire was part of a research initiative and that participation was entirely voluntary, with all responses collected anonymously.

To gain broader insight from more varied profiles, we also distributed the questionnaire publicly. To this end, we used LinkedIn, a platform designed for professional networking, as well as Facebook, a platform used both for recreational and professional purposes. We posted the call for participation on our personal LinkedIn and Facebook profiles (as our own network includes many individuals with substantial industrial experience), as well as on LinkedIn and Facebook pages and groups dedicated to software engineering professionals. The posts visibility was set as public so that other individuals who were interested in the study were able to further share them, which helped widening our reach. We wrote a brief article explaining the motivation behind our study, which served as a call for participation. Including the link to this article in our posts helped to attract more attention and increase engagement. We additionally circulated the questionnaire via email to professionals from the software industry and also asked them to share it within their own network so to broaden our reach beyond our immediate networks. Finally, we also promoted our survey in person during ICSE'24 and FSE'24 (as these two major SE conferences are attended by academics and professionals from industry), where we talked about our study to software practitioners attending the event and also handed out flyers with the link to the questionnaire. By leveraging on the above dissemination strategy we aimed to reach out to as many as possible different individuals with industrial experience in SE and solicit a large number of responses from participants all over the world, from different domains, backgrounds, and experience [12]. The first response was received on 16/02/2024 and the last response on 26/05/2024.

3.3 Result Analysis

The questionnaire includes two types of questions, namely multiple-choice and three open-text questions; thus we use a mixed-method approach where multiple-choice questions are analyzed quantitatively and open-text qualitatively as is common in similar research [5, 11]. Multiple-choice questions were quantitatively analyzed based on the number of responses chosen for each of the options provided. Some of these questions had an 'Other' option with an open-text box. The answers to those questions were merged into groups based on themes based on the same method we followed for open-text questions, described next. Open-text questions were assessed using thematic analysis [10], which involves searching for themes within a collection of textual data. We first analyzed the data rigorously and coded the responses to the questions. The codes are short descriptive tags of the responses that can then be clustered to form a theme map. An author of this paper conducted the analysis. The codes and themes were then independently evaluated by a second author, who suggested a more precise naming for two codes, agreeing with others[1].

4 Participant Demographic

In this section, we provide an overview of the demographics of the participants who completed our questionnaire.

4.1 Zühlke Questionnaire

The questionnaire within the Zühlke company was answered by 24 participants with highly skilled profiles. The majority of these participants (83.3%) have over 10 years of programming and professional experience and have worked in more than 3 companies throughout their career (79.2%). The participants were mainly software engineers/developers (62.5%), some were consultants (16.7%), as well as some participation from individuals with a managerial role such as CTO, Chief Engineer, etc. Since Zühlke offers software engineering solutions to various clients from different domains, the individuals participating in the questionnaire indicated 17 different domains when asked for the application domain of their work. Approximately 71% of the individuals indicated that they work on open source software projects within Zühlke.

[1]Unfortunately, release of raw data with codes is not possible due to ethics (see Acks).

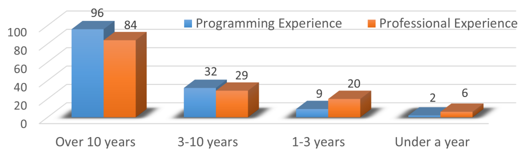

Figure 1: Participant past experience.

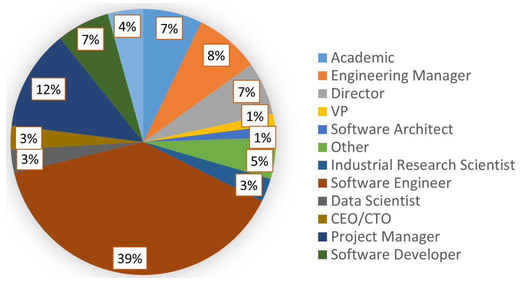

Figure 2: Participant roles (merged on role title for different levels of seniority).

4.2 Public Questionnaire

We initially received 139 responses to our questionnaire. However, 3 responses were excluded from the analysis, as the participants indicated that they had never worked in a professional software company. The remaining 136 respondents provided their consent and correctly answered the attention question.

Figure 1 shows the experience of the participants and their current roles. We observe that more than 60% of the participants have more than 10 years of professional experience in the software industry and around 70% have more than 10 years of programming experience. The participation of highly experienced individuals who have a deep understanding of software engineering practices based on many years of experience increases our confidence in gathering high-quality responses to our questionnaire.

The professional roles of the participants are presented in Figure 2. Note that for the sake of clear presentation, some of the roles that the participants entered were merged based on category, e.g., engineering manager also includes senior engineering manager. Overall, we can observe that the participants' experience relates to a wide variety of roles within the software industry. Most of the participants are software engineers (42%), project managers (13%) and engineering managers (8%), while others are in high managerial positions such as directors (7%), VPs (2%) or CEO/CTOs (2%). This helps ensure that the responses reflect the experience of professionals from all across the organizational chart.

Figure 3 presents the number of software companies that participants have worked for so far in their professional career. We can see that over 50% of the participants have worked in 3 or more software companies in their career. Furthermore, more than 80% of the participants have worked in at least 2 software companies. This increases our confidence that most of the responses received reflect the perspectives of individuals who have had varied experience within the software industry.

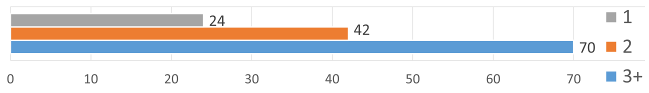

Figure 3: Number of participants that have been employed by 1, 2, or over 3 software companies throughout their career.

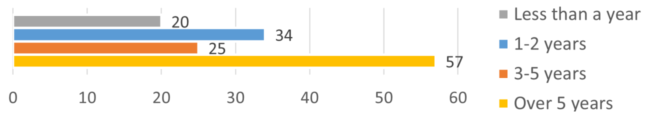

Figure 4: Number of years that the participants have been employed in their current/most recent company.

Figure 4 presents the number of years that a participant has been working at their current/most recent software company. More than 40% have been with the same company for over 5 years, indicating a strong familiarity with its practices.

Next, we investigate the profile of the companies for which the participants in our survey work for. In Figure 5, we present the application domain of the software on which our participants work. We can see that the participants come from a variety of backgrounds. This allows us to be able to generalize our findings across the software industry as we have representation from 10s of companies across a wide range of sectors. The figure also presents the size of the company the participants work for. We can see that we have the highest representation from large software companies with over 10K employees. This will be reflected in responses that represent well-established processes in line with industry standards around the hot fixing process. Additionally, we can see that the second largest company size category is companies with less than 100 employees. These are likely startups with possibly different hot fixing practices that fall in line with their different business needs. The bottom right plot in the figure presents the geographical area in which the participants' teams are located. We can see that most of the participants are from the Middle East, Europe, and the Americas. Finally, we ask the participants whether they work on open-source software as that could affect the hot fixing process. Our sample includes a mix of participants working on open-source as well as closed-source projects with the majority, 71.3%, working on proprietary, closed-source software.

To sum up, the profiles of the participants involved in this study are varied. Moreover, the profiles of the companies that they work at are also varied. This allows us to present results that can hold across the software industry. Our sample includes a large proportion of highly experienced individuals as well as individuals who are employed in well-established companies with over 10K employees. Thus, we can have high confidence that the responses received represent the industrial standards around hot fixing activities.

5 Findings

In this section, we present selected findings observed from the analysis of the responses to our questionnaire. All results are visually aggregated in the form of graphs, available in our artefact [1].

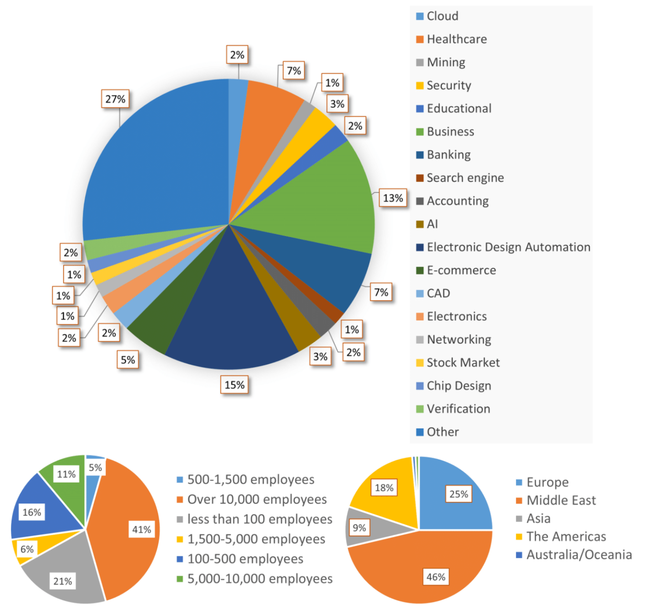

Figure 5: Profile information on the software companies that the participants work(ed) at most recently. We present the application domain of the software developed at the company (top), company size (bottom left), and geographical area where the participant's team is located (bottom right).

5.1 RQ1: Hot Fixing Terminology

We asked the participants about their familiarity with the terminology. First, we present the results from Zühlke. All participants indicated that they are familiar with the term "hot fix" with 83.3% saying that they clearly know what the term means. As for "hot patch", around 42% of the participants have never heard the term before and only 25% have a clear idea of what the term means. The responses pointed out differences in patch size, whether the change is temporary or permanent, whether it addresses a functional bug or other features. Regarding the definition of a critical bug, the majority of responses (54.2%) indicated that a critical bug would have high priority and high severity. Six participants (25%) explained that it depends on the context and that both metrics would be good indicators. When one or the other was chosen, was always given greater importance than priority when labeling a bug as critical (reported by 20.8%) with one participant explaining: *"Severity is well defined and understood and is a common base for defining critical bugs, failures. However, priority is indeterminate here, since it could be referring to development/mtce team assigned priority, operations (SLA) priority, or even client priority!".*

Next, we present the results of the public questionnaire (Figure 6). For "hot fix", over 70% of the participants had heard of the term before and thought that they had a clear idea of what it meant. As for "hot patch" only around 30% of the participants were confident with the meaning of the term whereas around 45% reported to not have heard of the term before. With regards to differences between terms 'hot patch' and 'hot fix' 5 participants, of 64 who responded, stated that they were unsure of what the difference was and 5

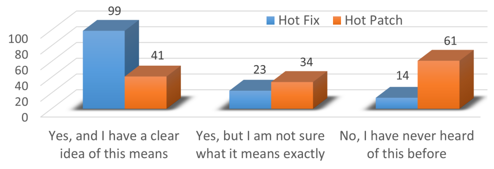

Figure 6: Have you heard the term before?

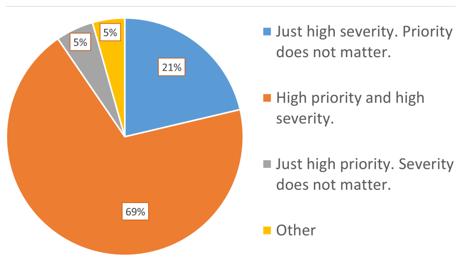

Figure 7: Responses to what classifies a bug as "critical" and thus would need hot fixing.

participants stated that they think there is no difference. As for the answers of the remaining 54 participants, we found 8 different themes for how they perceive the difference.

Firstly, 13 participants stated that a hot fix addresses a bug whereas a hot patch might also add features such as needed functionality that was previously overlooked. Other participants addressed differences in the size of the patch. However, the differences were inconsistent with 3 saying that a "hot fix" is larger and 5 saying the opposite. Another theme that emerged was differentiating between a code change and other changes. Here a hot fix would refer to a code or software change while a hot patch would refer to a data, hardware, usage, documentation change, etc. 8 participants suggested hot fixing refers to addressing time-critical issues while hot patching refers to techniques that patch software during runtime without a system reboot. One participant wrote: *"Hot Fix: A fix required for an issue that needs urgent attention (..) usually done on a production environment without following the right software/agile/scrum release lifecycle, a hot patch is an update that may not necessarily be linked to a critical issue, and one that does not require an update of the application or software to implement the patch."* 6 participants commented on the permanence of the code change. The next two themes pertain to the users of the software: whether the change is user-facing and whether the change would be rolled out to all users or just a subset. Other comments were that hot fixing does not follow traditional development flow (e.g. skips testing) whereas a hot patch would and that hot fixes would be immediate changes and hot patches would be scheduled.

The participants input on the criticality of a bug is presented in Figure 7. 69% of the participants regard a critical bug as one that has both high severity and high priority. 21% regard severity as a better indicator than priority for the criticality of an observed bug.

RQ1: Terminology around hot fixing among professionals remains inconsistent, but clear themes emerge when discussing it. Both *priority* and *severity* are indicators for how critical a bug is (69% of participants deem both equally important), with more importance given to its severity (21% of the participants).

5.2 RQ2: Hot Fixing Current Practices

We present results pertaining to the current industrial practices for hot fixing. First, we delve into the results from Zühlke. Due to limited space, we present the results of what we deemed to be most interesting. The full results can be found in our artefact [1].

Generally, it seems that hot fixes at Zühlke are rolled out within hours on average (answered by 58.3% questionnaire participants) in contrast to the public questionnaire in which it was most commonly reported that rollout takes a few days on average (answered by 37.5%). Each hot fix involves a small number of employees usually involving the team lead and the developer on call/one maintaining the code. Most commonly, the need for a hot fix is identified through a report by a customer/client. The team lead is made aware of the issue and then involves the relevant software developer. The software developer creates the hot fix in a separate branch, tests it, and after a peer-review is then deployed. One of the participants comments: *"Communication & collaboration is most important! As is treating every commit as production-ready, so that if we need to hot fix, we don't need to cherry pick from other work".*

It is mentioned that ideally, hot fixing should be no different than a normal release (such that it follows the eight steps mentioned in Section 6.3). One participant clarifies: *"Defect raised by support, or ops/monitoring, or product team. Item is triaged and given severity and priority and in this situation is also flagged as requiring a hotfix [rare]. Software Development Manager is informed of need for a hotfix and takes ownership. Assigns task to a dev+tester. They work the problem calling in whoever they need, and then go through an expedited dev and test process with a higher priority than any in progress work. Status updates are communicated by Dev Manager to Support/ProdMan as necessary so customers can be informed. Once rollout has occurred, support/prod man are informed of status so they can communicate to customers. Root Cause Analysis then begins to understand why the hotfix was necessary to begin with.".* However, that is simply the ideal situation. One participant mentions that *"sometimes patch pipeline may skip some environments to avoid contention".* Another mentions the overhead of formal releases which might result in beta releases for affected customers in the meantime.

When asked about the categories of the critical issues that hot fixes target we got varied responses. Some mentioned that in a technical sense, these issues are varied but what makes them similar is their big business impact. Others mentioned that they are most commonly a result of gaps in testing which most commonly cause issues in user journeys and the user interface (caused by software bugs, data issues, or system issues such as networking and storage).

We now present the results of the public questionnaire. We first investigate the frequency of hot fixing within software companies. The results are presented in Figure 8. When looking at estimates for the average number of hot fixes that professionals observe or contribute to within their company, we can see that the numbers are low. Most commonly, they observe or contribute to between 1 to 5 hot fixes each month. This implies that hot fixes are exceptional cases and much rarer than regular bug fixes.

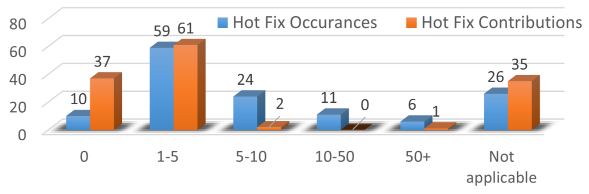

Figure 8: Participant responses for how many hot fixes they observe or contribute to on average monthly.

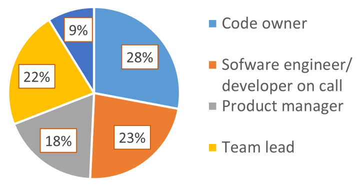

Figure 9: Participant responses for who on the team is mainly responsible for handling the hot fix.

We asked participants who on their team is responsible for handling hot fixes, with results shown in Figure 9. Clear ownership is crucial, as a lack of it can compromise productivity. We aimed to see if responses were consistent across participants from different companies, domains, and roles. The most common answers were: the owner or maintainer of the affected code (28%), the on-call engineer (23%), the relevant team lead (22%), and the product manager (18%). A few participants gave alternative responses. For example, one noted that the on-call engineer typically handles it by rolling back, rather than applying a hot fix. However, rollbacks are only effective if the defect was introduced in the latest release. This isn't always the case, as issues can stem from changes in the software environment, such as unexpected workload spikes, hardware failures, or long-dormant bugs that have only recently surfaced. Another person mentioned that it would be a hybrid of people: *"A hybrid of the engineer on call and the owner of the affected code. Responsibility starts from on call and goes down to maintainer (all engineers)"*.

Figure 10 illustrates participants' perspectives on whether hot fixes typically target specific categories of bugs. Two thirds of the participants responded that they think this is not the case. The remaining third that believed categories exist or might exist were asked to briefly describe these categories using open text. We found that the participants' answers could be grouped into two categories: technical causes and observable outcomes.

The participants touched on the following 10 technical causes of issues that hot fixes target: requirements, configurations/versioning, security, performance, data issues, compatibility/integration, GUI issue/interface, license issue/regulation, code logic/concurrency issues, and database issues. Answers also included observable outcomes that warrant the need for hot fixing. These are non-technical categories of issue types but rather critical situations where a hot fix would be needed: crashes, customer requests, and business needs/affects service-level agreement (SLA)/ affects market.

To understand the urgency of issuing a hot fix, we asked participants how long it typically takes, on average, from detecting the need for a hot fix to its deployment. We present the results in Figure 11. Surprisingly, 37.5% of the participants responded that this process takes a few days on average. The time is clearly affected by the type of system (e.g. safety critical), the severity of the issue, and the management style of the project. The next most common response, aligning with what we expected, was a few hours.

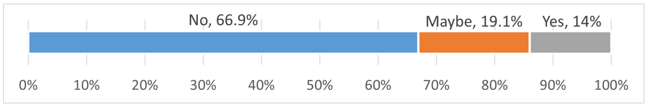

Figure 10: Do hot fixes fall in specific bug categories?

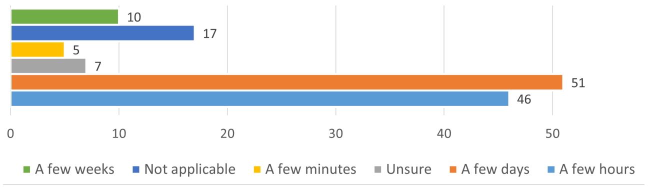

Figure 11: Average time from detecting the need for a hot fix until the hot fix is deployed.

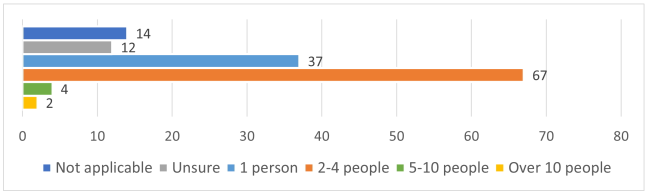

Figure 12: Number of people usually involved in the hot fix.

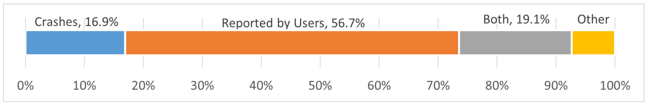

Figure 13: Mode of hot fix discovery.

Figure 12 presents the average number of people involved in the hot fixing pipeline from start to finish. 49.3% of participants responded that between 2 and 4 individuals are involved in the hot fix process and 27.2% participants responded that it is just one individual that is solely responsible for the hot fix. From here, we can see that the majority of the responses indicate that a very small number of people are responsible for the hot fix when it is needed.

We also investigate what the most common way that the need for a hot fix is discovered. The results are presented in Figure 13. Around 93% of participants responded that hot fixes target issues identified through crashes or reports made by the users of the system. One participant detailed that specifically, crashes are identified through systems such as Sentry [28] and reports by users are aggregated through customer success managers. The vast majority of the responses nearing 57%, claimed that their need is directly reported by the users. Other participants indicated that identification is done through system monitoring and health checks or security bulletins.

Finally, we asked the participants to succinctly describe the hot fixing pipeline in their company. We received informative and detailed responses from 54.5% of the participants which sheds light on current industry standards for hot fixing. Three participants explained that within their company there is no one single process or pipeline for hot fixing that they know of. They claim that these situations are handled on a case-by-case basis depending on the type of issue and its effect. The rest of the responses detail such a pipeline. The responses received highlighted different parts of the process which we aggregated to present the full hot fixing pipeline. The participants detailed 8 different steps: discover, report, triage, fix, test, deploy, verify, and post-mortem.

First, the critical issue is **discovered**. It can be discovered in multiple ways. One option is internally by internal users, testers, or someone from the development team. Another option is externally by a stakeholder or customer. In this case, the issue might need to be validated on the customer side to confirm that it is a real critical issue. A third option is through metrics, crashes, or automated error monitoring (e.g., through systems such as Sentry [28]).

Second, the issue is **reported**. The issue can be raised as a support ticket on well-known platforms such as Jira [7] (which can be monitored live), designated Slack [30] channel (can also support manual or automated tickets), etc. As these issues are time-sensitive, in some cases, they may be raised directly to the relevant individuals who will foresee the hot fix to save time. If a specific person who needs to handle the fix is not in working hours, they may be contacted to help (in this case, most often a rollback would be more likely than an actual fix).

Third, the alerted relevant individuals can then **triage** the issue and identify its root cause. This is normally done by the development/product team. The decision for how to proceed can either be done by multiple people (one person mentioned that engineers would huddle to take this decision together) or by the team lead/product owner/manager. One person mentioned that hot fixes would have to go through the shiproom [4]. It is also common for an ETA to be provided for the fix.

Fourth, the individual(s) responsible can begin **remediation** of the issue. Current tasks are paused, and the focus is shifted to the fix. Different workarounds are considered to optimize the limited time. If multiple individuals are involved, they may split the work so that one person reproduces the issue by creating tests, another examines the logs, and so on. The fix is generally done on a separate branch and a pull request is generated in the end.

Fifth, the issue is **tested** in preproduction using unit/integration tests. Regression and unit tests must be added to ensure that the fix works as expected. The CI/CD pipeline offers a second layer of verification in the development environments. The pull request passes a code review process by peers and generally must be signed off by at least one engineer. In this case, a final approval is usually required by someone in a managerial position.

Sixth, the issue is **deployed**. Normally, a short description of the update is generated to accompany the fix. There are many ways that deployment seems to be done depending on the scenario. A manager generally makes the call for the most appropriate course of action. In some companies, this needs to be coordinated with a designated deployment team, who sets a deployment procedure and tests this procedure. In other companies, the process is much more laid back, and automated pipelines create a build with the fix and deployment carries out via the CI/CD pipeline. Historically, hot fixing procedures often relied on techniques like bytecode weaving [8] and monkey patching [18] to implement live patching. However, due to the inherent risks and potential instability associated with these live methods, modern enterprises have largely moved away from them when addressing critical issues.

Depending on the type of fix and its urgency, the hot fix will be deployed immediately or bundled with the next release. In the latter case, a unique build containing the hot fix may be sent directly to the target customer. Field engineers will take action to identify affected customers and share the new build.

Seventh, the issue needs to be **verified** on customer environments. Post-deployment QA processes will take place and the fix is monitored. When necessary, communication about the situation will need to be opened with the stakeholders.

Eighth, **post-mortem** activities can start. This includes various analyses to review what happened and why it happened. These reviews normally include what can be done to prevent similar issues from arising in the future.

The survey's participants included other interesting comments in their answers. In certain companies, strict permitting processes are in place for emergency release into production. This suggests that the caliber of the criticality of an issue must be high and that the generated fix would likely need to be heavily tested. Someone mentioned that once the team of people who will handle the hot fix is identified, they will remain delegated to the hot fix until it is remediated, even if this comes at the expense of other delays. This highlights that hot fixing is the highest priority activity. Another person mentioned designated firefighters that handle such issues.

RQ2: While the specifics of hot fixing (frequency, responsible individuals, etc.) differ from one company to another, the general pipeline for hot fixing remains similar across the responses received, encompassing subsets of the following stages: discover-report-triage-fix-test-deploy-verify-post-mortem.

5.3 RQ3: Hot Fixing Tooling and Automation

We present the participant responses on the state of current tooling within their current/most recent company in Figure 14.

We begin by presenting the results within Zühlke. The responses indicate that the latter stages (verification and deployment) utilize automated tooling whereas the critical bug detection and hot fix generation is more manual. Interestingly, the responses vary among the different participants within the same company hinting that not all employees are aware of the available tooling or that the tooling is specialized for specific use cases within specific teams.

Next, we analyse the responses to the public questionnaire. Similarly to the Zühlke's results, these responses also point out that the latter stages in the pipeline are more automated while, tooling for detecting the need for a hot fix is less common. Only 13.2% of participants were able to assert with certainty that this type of automation exists within their company, while 48.5% responded that this type of tooling likely does not exist within their company. In terms of automating the creation of the hot fix, this is the stage of the pipeline with the least advanced tooling. In fact 65% of the

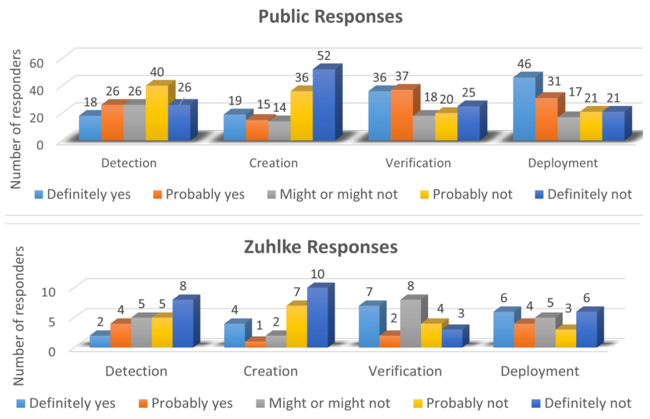

Figure 14: Participant responses for existing tooling within their current/most recent company.

respondents reported that tooling for automatically generating hot fixes does not exist. As for verification once the hot fix is generated, the responses indicate that parts of this process are automated. This is likely because the existing tooling for verifying more general-purpose bug repair also applies to verifying hot fixes. Finally, the deployment of the hot fix. This is the step that 33.8% of the participants indicated is definitely automated within the pipeline, the most of any of the other steps. Similarly to the verification step, this might be because all patches are deployed the same way and no specialized tooling is required for hot fixing specifically.

RQ3: More automated tooling exists for the latter stages of the hot fixing pipeline (53.7% of the questionnaire participants indicated that automated solutions probably or definitely exist to verify the hot fix and 56.7% indicated such solutions exist to deploy hot fixes) than for the earlier stages (32.3% participants indicated such techniques for hot fix detection and 25% for fix generation).

5.4 RQ4: Standardization of Hot Fixing

We assess hot fixing standardization from two angles: company-level and individual participant profiles. To avoid bias, only responses from the public questionnaire were used, excluding those from Zühlke participants.

Company Profile and Hot Fixing Practices: The first measure we looked at was company size. We compared the two ends of the range we specified in the questionnaire which are companies with over 10K employees and those with under 100 employees. We had 56 responses from participants in companies with over 10K employees and 29 responses from companies with under 100 employees. Regarding the frequency of hot fixing, more hot fixes are observed in small companies than larger ones. 69.7% of participants from large companies responded that they observe at least one hot fix a month whereas in smaller companies 79.3% responded this. We also find that larger companies have more automated tooling for hot fixing. We looked at the "probably yes" and "definitely yes" answers to the question of whether tooling exists for a given hot fixing step. The percentages were always higher for larger companies with over 10K employees than smaller companies of 100 employees (41.1% vs. 27.6% for detection, 33.9% vs. 13.8% for creation, 67.9% vs. 37.9% for verification, 58.9% vs. 51.7% for deployment).

Next, we looked into patterns in companies with similar application domains. We looked at four different categories: chip design and its verification, banking/business/fintech, AI, and security/defense. When looking at the average time for a hot fix to be deployed from the time of detection we found differences across the categories. Hot fixes are developed slowest for hardware-related projects (semiconductors, electronic design automation, etc.) with 20% of participants responding that the average is a few weeks and 63.3% responding that it takes a few days. As for banking, it seems that the industry standard is a few hours (55.2%) or a few days (41.4%). For AI and security, hot fixes are developed fastest, most commonly within a few hours. As for the chip design category 20.6% of the participants indicated that between 10-50 hot fixes are observed monthly. For the other categories, this number was much less such that for AI and security all participants responded that the average is only between 1-5, for banking only 3% responded that more than 5 hot fixes are observed monthly on average.

Finally, we investigated possible differences in the type of software being developed: open-source vs. closed-source. We found that when the software is open source, 80% of the participants answered that a customer reports the need for a hot fix (as opposed to 56. 7% observed from the total of the participants in Section 6.3).

Participant Profile and Hot Fixing Practices: We investigated for patterns in the responses given by the participants based on their professional experience and current role.

First, we grouped the results based on the professional experience of the participants. We compared two groups of participants: experienced professionals with over 10 years of experience (84 participants) and less experienced ones with less than 3 years of experience (26 participants). When looking at their familiarity with terminology, experience played an important role. For the more experienced group, 90.1% of them were familiar with the term "hot fix" and 64.2% with the term "hot patch" in comparison with 88.5% and 42.3% for the less experienced group.

Next, we grouped participants with managerial roles (such as team leads, directors, product/software/project managers) and ones with a non-managerial role (such as data scientists, software developers, production engineers). From the ones who report observing hot fixes within their company, we looked at how many participants contribute to hot fixing within the two groups. In the non-managerial group, 62.1% of the participants reported contributing to hot fixing processes within their company, whereas 81.5% of the ones in the managerial group reported doing so. Thus, a manager is highly likely to be involved in hot fixing whereas individuals in non-managerial roles are likely only involved in specific scenarios.

RQ4: Different factors affect the hot fixing practices within industry, e.g., larger companies have more automated tooling; hardware-related companies have the slowest hot fix deployment time, while AI and security have the fastest; managers are more likely to actively participate in the hot fix pipeline.

6 Discussion

In this section, we discuss key observations and implications of our study for both researchers and practitioners.

6.1 Key Observations

Terimonology: The terminology for what constitutes a hot fix is inconsistent in the academic literature [16]. What we have found is that the case is similar among people in the software industry. However, the definitions gathered from practitioners fall into distinct categories and thus are more streamlined. Consistent and clear terminology is essential. It allows for the body of work to be more cohesive, for processes to be more productive, and for better collaboration between the different communities. We find that individuals with more experience are more familiar with the terminology. As such, we urge training on this for new hires. We plan to explore this further through developer interviews to gain a deeper understanding of why this occurs, its practical implications, and to develop a standardized framework.

Current practice: We outline the pipeline for hot fixing based on the questionnaire responses received from the software professionals. Collectively, it consists of 8 steps. These steps do not seem to differ too much from the steps taken for traditional bug repair (discover-report-triage-fix-test-deploy-verify-post-mortem). At first glance, this seems to be contrary to our original assumption that hot fixing would require skipping certain steps of the bug fixing process to improve productivity and meet the time-criticality requirements. However, this is not necessarily the case. The identified steps were aggregated based on participant input, with no single respondent including all eight steps in their answers. Instead, companies tend to skip different phases to save time. For example, some companies avoid the reporting step via issue management platforms and instead contact the relevant individuals to handle the fix directly, other companies skip extensive testing. As a Zühlke employee noted, an ideal hot fix would follow all regular bug fixing steps, but this is often impractical in real-world scenarios.

The deployment process is more intricate in some companies than in others depending on the management style that they follow. Companies that follow strict release timelines and traditionally require intricate deployment planning from a release time have a harder time with hot fixing. Answers from practitioners who work in such companies explained that for a hot fix to be released, it needs to be signed off by specific individuals and a deployment plan needs to be put in place and tested. Whereas individuals from other companies would directly deploy after quick testing in the development environment. As such, we can conclude that while similar steps are followed for hot fixing between different companies (borrowed from traditional program repair), different companies hot fix differently and optimize their processes by skipping different steps from the regular bug repair pipeline. This can be taken a step further in saying that processes are different not only across companies but also across different teams within the same company as we did observe variability in the responses of the Zühlke internal case study. As it stands, there is no industry standard that can be considered the state-of-the-art model for hot fixing software.

Automation: Tooling for verification and deployment is further developed than the creation of the hot fix, for example. We assume that is because tooling for verification and deployment can be used as is for the purpose of hot fixing whereas for the creation of the hot fix traditional program repair is currently too slow and unreliable [17] to be productive. More curated tooling can be developed for this specific purpose taking into account our finding that the number of involved in a specific hot fix is small (4 or less).

6.2 Implications for Researchers

Use more consistent terminology: We urge researchers to use consistent terminology when referring to the same software engineering activity or phenomenon. This helps create a community and unify the shared knowledge and body of work around the topic. We hope this makes work more discoverable, fostering collaboration and enabling advancements in hot fix innovation.

Conduct more empirical studies: More empirical research is needed to identify the tooling practitioners need most. As expected, software architectures, the choice of languages and technology stacks, development tooling, and consequently, development processes vary significantly across projects and between different companies. This variability means that the specifics of how features are developed and bugs are fixed differ, and will likely always differ. These differences are influenced by factors such as the domain, regulatory requirements, and client preferences and needs. Consequently, the hot-fixing process and its associated tooling will inherently vary across clients and application domains. More in-depth analyses are needed to understand the specific needs of the individuals at the time a hot fix is needed. Our findings show that typically only a small group of people (only 4% of participants claimed more than 5) is involved in hot fixing with roles varying across companies. Therefore, understanding their collaboration more intricately is key to developing effective, tailored tooling.

Direct research toward generation: Given the results of the questionnaire, tooling for the step of the hot fix pipeline that involves the generation of the actual fix is the least developed. Thus, we urge the community to direct research efforts towards efficient automated program repair that can be utilized for the purpose of hot fixing rather than the other steps of the process.

6.3 Implication for Practitioners

Adopt modern software development processes and automation: Many participants noted the benefits of using verification and deployment tools for hot fixing in their companies. Practitioners in organizations without such automation could consider implementing it to enhance their own processes. Participants at Zühlke reported shorter times to implement a hot fix compared to the public survey average. This efficiency is attributed to their practice of automating routine tasks such as builds, regression testing, and deployment. In some projects, Zühlke teams release daily or even multiple times per day into production. In these environments, processes and tooling are designed to handle hot fixes seamlessly, treating them as a natural part of the workflow. This demonstrates that organizations can benefit greatly from adopting modern development practices that enable frequent, low-cost releases [24]. By doing so, they can manage hot fixes at a lower cost, avoiding unnecessary stress or disruptions. It is important to note that this is not always possible and depends on the context and application domain. For example, industries such as banking, healthcare, and physical systems often prioritize caution in deployments due to regulatory requirements, critical reliability needs, and risk aversion, which can limit their ability to fully embrace rapid and iterative agile processes. Even in urgent cases, deviations from the standard bug-fixing process can be minimized if organizations maintain a single, high-quality deployment pipeline that supports various scenarios. Rather than creating a separate hot fix pipeline, the same pipeline can handle all changes, including urgent ones, by incorporating configurable gates and approval processes tailored to the context. This configurable pipeline aligns with our RQ2 findings , where participants described a traditional bug fixing process that omits certain steps depending on the urgency of the issue.

Transparency: We urge software companies to be more transparent about their internal software engineering processes. While confidentiality can pose limitations, it can accelerate innovation in this area and allow for improved processes for the industry without necessarily compromising intellectual property. The responses we received in this study were very detailed and proved that transparency is indeed possible, it just needs to be prioritized.

Preempt hot fixing whenever possible: We found that most hot fixes are issues reported by users. We urge companies to put additional measures in place to avoid and preempt this. This can be by having systems for metric analysis, specialized defect prediction, etc. The optimal hot fixing scenario is avoiding the need for a hot fix. Thus, we think that there is value in understanding the types of issues that result in hot fixes and dedicate resources to avoiding their occurrence or discovering them before reaching the end user.

7 Threats to Validity

The design of the study and the selection of questions that were included in the questionnaire may add bias. We mitigated this by conducting questionnaire testing before its public dissemination. The questionnaire was answered by 24+136 individuals which is considered a fairly large number in comparison with similar work [11, 14, 32]. However, responses still may not represent the software industry as a whole. We mitigate this threat by adding profile questions to survey the expertise and background of these individuals. As shown, many participants were highly experienced and well-positioned to offer deep insights into current industry standards. Since no incentives were offered, participation was driven by genuine interest. All participants correctly answered the mid-questionnaire attention check, reinforcing data validity and indicating strong engagement.To mitigate validity threats from demographic imbalances, RQ4 only analyzes groups with sufficient data.

8 Related Work

Survey-based research is a method that aims to gather data from a large population of interest [20, 22, 27]. Using questionnaires to gather insights directly from relevant stakeholders is an effective approach to better understand real-world practices [20, 27]. Such a technique is useful in gaining a better understanding around software engineering matters and shortening the gap between the academic and industrial communities [19, 27, 34]. Online questionnaires have been used to understand a wide range of different software engineering activities and phenomena within the industry, covering both technical (e.g. [21, 31, 32, 34, 35]) and socio-technical aspects (e.g. [6, 11, 14, 25, 29, 33]). Since our work is the first survey on hot fixing in practice, in this section we focus on existing survey-based research carried out on automated program repair and runtime anomalies, which are the closest topics to hot fixing.

Questionnaires have been employed to learn about practitioners' perspectives on APR [21, 35, 36], on how trust in automated repair patches can be enhanced [25], and to inform the deployment of user-centric APR tools [34]. Contrary to this previous work, our study focuses on hot fixing specifically and follows the full pipeline of this process instead of just the patch generation step. Steidl et al. [31] conducted interviews with practitioners to understand how runtime anomalies are tackled, which is particularly relevant to our work as these runtime anomalies might need to be hot fixed and thus would induce similar processes. However, their study predominantly focuses on anomaly detection, specifically, detection through runtime monitoring data. Our study is the first to investigate hot fixing practices specifically.

While the collective knowledge around the software engineering of hot fixing from an academic perspective has been recently reviewed [15, 16], our work is the first to investigate industrial perspectives, offering an in-depth analysis of how hot fixing is approached in real-world environments.

9 Conclusions and Future Work

We investigated hot fixing practices within the industry by disseminating a dedicated online questionnaire. We collected responses from 24 Zühlke employees, and further 136 software professionals from various companies, domains, roles, and regions. The analysis of the responses unveiled a mixed opinion regarding the definitions of hot fix and hot patch, thus confirming the inconsistency in terminology used in the academic literature [16]. We found that while there are general guidelines for industry standards on what a hot fix pipeline should be (such as involving a small number of people and delivering the hot fix within a few days or less), there remains considerable variation and less alignment than expected. This diversity presents both challenges and opportunities for refining and improving hot fix practices. We further found practitioners most commonly utilize automated verification and deployment for hot fixes while tools for hot fix generation are less common, highlighting a potential area for growth. We plan to conduct interviews to gain an even deeper understanding of the emerged themes.

Acknowledgments

We thank all participants for their insights, Zühlke Engineering Ltd for their collaboration, and Mark Harman for his mentorship. This study was approved by UCL CS Ethics Committee (UCL/CSREC/R/43).

References

[1] Github - carolhanna01/hotfixindustrialviews. https://github.com/carolhanna01/HotFixIndustrialViews. Accessed: 2024-01-14.
[2] No man's sky. https://www.nomanssky.com/. Accessed: 2024-10-05.
[3] To hotfix or to rollback? that is the question — mike madison. https://mikemadison.net/blog/2022/01/03/to-hotfix-or-to-rollback-that-is-the-question. Accessed: 2024-10-05.
[4] Navigating the waters of software releases: The shiproom meeting — behrouz.nl. https://www.behrouz.nl/article/navigating-software-releases-shiproom-meeting, 2024.

[5] Al-Subaihin, A. A., Sarro, F., Black, S., Capra, L., and Harman, M. App store effects on software engineering practices. *IEEE Transactions on Software Engineering 47* (2 2021), 300–319.

[6] Al-Subaihin, A. A., Sarro, F., Black, S., Capra, L., and Harman, M. App store effects on software engineering practices. *IEEE Transactions on Software Engineering 47*, 2 (2021), 300–319.

[7] Atlassian. Jira | issue & project tracking software | atlassian. https://www.atlassian.com/software/jira, 2024. Accessed: 2024-10-05.

[8] Baker, J., and Hsieh, W. Runtime aspect weaving through metaprogramming. 86–95.

[9] BBC. Chaos persists as it outage could take time to fix, says cybersecurity firm boss - bbc news. https://www.bbc.co.uk/news/live/cnk4jdwp49et, 2024. Accessed: 2024-10-05.

[10] Braun, V., and Clarke, V. Thematic analysis. *APA handbook of research methods in psychology, Vol 2: Research designs: Quantitative, qualitative, neuropsychological, and biological.* (3 2012), 57–71.

[11] Davila, N., Wiese, I., Steinmacher, I., Silva, L. L. D., Kawamoto, A., Favaro, G. J. P., and Nunes, I. An industry case study on adoption of ai-based programming assistants. *ICSE-SEIP '24: Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice* (4 2024), 92–102.

[12] Galster, M., and Tofan, D. Exploring web advertising to attract industry professionals for software engineering surveys. In *Proceedings of the 2nd International Workshop on Conducting Empirical Studies in Industry* (New York, NY, USA, 2014), CESI 2014, Association for Computing Machinery, p. 5–8.

[13] George, D. A. S. When trust fails: Examining systemic risk in the digital economy from the 2024 crowdstrike outage. *Partners Universal Multidisciplinary Research Journal 1* (7 2024), 134–152.

[14] Goncalves, P. W., Goncalves, J. S., and Bacchelli, A. Constructive code review: Managing the impact of interpersonal conflicts in practice. *ACM International Conference Proceeding Series* (4 2024), 334–345.

[15] Hanna, C., Clark, D., Sarro, F., and Petke, J. Hot fixing software: A comprehensive review of terminology, techniques, and applications. https://arxiv.org/abs/2401.09275, 2024.

[16] Hanna, C., and Petke, J. Hot patching hot fixes: Reflection and perspectives. *Proceedings - 2023 38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023* (2023), 1781–1786.

[17] Harman, M. Scaling genetic improvement and automated program repair. *Proceedings - International Workshop on Automated Program Repair, APR 2022* (2022), 1–7.

[18] Hunt, J. *Monkey Patching.* Springer International Publishing, Cham, 2023, pp. 487–490.

[19] Hynninen, T., and Jantunen, S. Questionnaire approach for assessing software engineering and quality assurance practices. *2022 45th Jubilee International Convention on Information, Communication and Electronic Technology, MIPRO 2022 - Proceedings* (2022), 1301–1306.

[20] Kitchenham, B., and Pfleeger, S. Principles of survey research part 1: turning lemons into lemonade. *ACM SIGSOFT Software Engineering Notes 26*, 6 (2001), 16–18.

[21] Meem, F. N., Smith, J., and Johnson, B. Exploring experiences with automated program repair in practice. *Proceedings - International Conference on Software Engineering* (2024), 1047–1057.

[22] Molléri, J. S., Petersen, K., and Mendes, E. An empirically evaluated checklist for surveys in software engineering. *Information and Software Technology 119* (2020), 106240.

[23] Morris, J. C., and Mayer, M. K. Canaries in coal mines and normal accidents: The crowdstrike outage and its lessons for critical infrastructure. *Journal of Critical Infrastructure Policy* (2024).

[24] Nicole Forsgren, PhD, J. H. G. K. *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations.* IT Revolution, 2018.

[25] Noller, Y., Shariffdeen, R., Gao, X., and Roychoudhury, A. Trust enhancement issues in program repair. *Proceedings - International Conference on Software Engineering 2022-May* (2022), 2228–2240.

[26] Ogundipe, O., and Aweto, T. The shaky foundation of global technology: A case study of the 2024 crowdstrike outage. *Article in International Journal of Multidisciplinary Research and Growth Evaluation* (2024).

[27] Punter, T., Ciolkowski, M., Freimut, B., and John, I. Conducting on-line surveys in software engineering. In *2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings.* (2003), pp. 80–88.

[28] Sentry. Application performance monitoring & error tracking software | sentry. https://sentry.io/welcome/, 2024. Accessed: 2024-10-05.

[29] Sesari, E., Sarro, F., and Rastogi, A. It is giving major satisfaction: Why fairness matters for developers, 2024.

[30] Slack. Ai work management and productivity tools | slack. https://slack.com/intl/en-gb/, 2024. Accessed: 2024-10-05.

[31] Steidl, M., Dornauer, B., Felderer, M., Ramler, R., Racasan, M.-C., and Gattringer, M. How industry tackles anomalies during runtime: Approaches and key monitoring parameters.

[32] Stevenson, J., and Wood, M. Recognising object-oriented software design quality: a practitioner-based questionnaire survey. *Software Quality Journal 26* (6 2018), 321–365.

[33] Storey, M.-A., Zimmermann, T., Bird, C., Czerwonka, J., Murphy, B., and Kalliamvakou, E. Towards a theory of software developer job satisfaction and perceived productivity. *IEEE Transactions on Software Engineering 47*, 10 (2021), 2125–2142.

[34] Williams, D., Callan, J., Kirbas, S., Mechtaev, S., Petke, J., Prideaux-Ghee, T., and Sarro, F. User-centric deployment of automated program repair at bloomberg. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice* (New York, NY, USA, 2024), ICSE-SEIP '24, Association for Computing Machinery, p. 81–91.

[35] Winter, E., Bowes, D., Counsell, S., Hall, T., Haraldsson, S., Nowack, V., and Woodward, J. How do developers really feel about bug fixing? directions for automatic program repair. *IEEE Transactions on Software Engineering 49* (4 2023), 1823–1841.

[36] Zhang, Q., Zhao, Y., Sun, W., Fang, C., Wang, Z., and Zhang, L. Program repair: Automated vs. manual. *arXiv.org* (2022).

[37] Zoysa, S. D. Microsoft global outages caused by crowdstrike software glitch.