

# Exploring Experiences with Automated Program Repair in Practice

Fairuz Nawer Meem\*  
fmeem@gmu.edu  
George Mason University  
Virginia, USA

Justin Smith†  
smithjus@lafayette.edu  
Lafayette College  
Pennsylvania, USA

Brittany Johnson‡  
johnsonb@gmu.edu  
George Mason University  
Virginia, USA

## ABSTRACT

Automated Program Repair, also known as APR, is an approach for automatically repairing software faults. There is a large amount of research on automated program repair, but very little offers in-depth insights into how practitioners think about and employ APR in practice. To learn more about practitioners' perspectives and experiences with current APR tools and techniques, we administered a survey, which received valid responses from 331 software practitioners. We analyzed survey responses to gain insights regarding factors that correlate with APR awareness, experience, and use. We established a strong correlation between APR awareness and tool use and attributes including job position, company size, total coding experience, and preferred language of software practitioners. We also found that practitioners are using other forms of support, such as co-workers and ChatGPT, more frequently than APR tools when fixing software defects. We learned about the drawbacks that practitioners encounter while utilizing the existing APR tools and the impact that each drawback has on their practice. Our findings provide implications for research and practice centered on adoption and use of APR.

## KEYWORDS

automated program repair, software bugs, software tools

### ACM Reference Format:

Fairuz Nawer Meem, Justin Smith, and Brittany Johnson. 2018. Exploring Experiences with Automated Program Repair in Practice. In *Proceedings of 46th International Conference on Software Engineering (ICSE 2024)*. ACM, New York, NY, USA, 11 pages. <https://doi.org/XXXXXXX.XXXXXXX>

## 1 INTRODUCTION

Software practitioners are susceptible to making mistakes. These “mistakes” manifest as software defects that can affect the software’s performance and usability [39]. Software defects can have a significant impact on numerous sectors. For example, over 10,000 patients in the United Kingdom’s NHS (National Health Service) were at risk of receiving the wrong medication due to a software flaw that was found in 2018. Software design flaws were to blame for the Boeing 737 Max tragedy that occurred in March 2019 [2]. Software defects,

as previously stated, not only result in monetary losses but also in the loss of employment, even lives. These software issues can be expensive and laborious [5] [33] as some bug types are simply challenging to test for in an exhaustive, predictable manner [27].

Therefore, an important and integral part of the software engineering process is finding and fixing software defects. The strain of manually correcting the growing number of programming errors can be significantly reduced by automated program repair [10]. As a result, automated program repair, which aims to reduce the debugging effort by automatically generating a fix for the faulty program [9] [24] [22] [12] [13], has been an active research subject in the software community [45]. While there exists numerous tools that support finding and fixing defects, research has shown that developers may not always be receiving adequate tool support for fixing defects [4, 14, 40]. Further, despite the wealth of research that exists on the feasibility and quality of automated program repair tools and techniques, few studies have explored the perspective of those who are or could be using these tools.

To better understand and work towards improving APR tool use in practice, we conducted an empirical study where we surveyed software practitioners on their experiences with and perceptions of existing APR tools and techniques. We asked practitioners questions regarding their APR tool use in the workplace, challenges they encounter when using APR tools, and their thoughts on the ideal automated program repair tool. Based on responses from software practitioners, including software team managers, developers, and QA specialists, we established that demographics like job position, company size, total coding experience and preferred language have significant relationships with awareness of and experiences with APR. We also collected and analyzed attitude towards APR tools currently in use and the human-centric challenges faced by the developers that restrict their use of APR tools.

In our paper, Section 2 outlines related existing work. Section 3 describes our survey design, survey respondents, and analysis. In Section 4, we discuss findings from our survey responses and limitations of our study in Section 5. Finally, in Section 6, we discuss the implications of our work.

## 2 RELATED WORK

APR tools support the ability to automatically repair a fault, reducing the cost of repairing software defects [10, 36] and error-proneness of manual fixes [30]. As such, the field of automatic program repair has seen a lot of progress over the years. Many scholars have attempted to concentrate their efforts over the years on helping developers and testers with the challenging and time-consuming process of addressing and fixing bugs. The views of

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

ICSE 2024, April 2024, Lisbon, Portugal

© 2018 Association for Computing Machinery.

ACM ISBN 978-1-4503-XXXX-X/18/06...\$15.00

<https://doi.org/XXXXXXX.XXXXXXX>

those who use APR are essential for producing more accurate outcomes in this field of study. In APR studies, software engineers are frequently mentioned but rarely directly engaged. It is necessary to have a more thorough grasp of the human aspects of APR adoption and use. Without this knowledge, it will be challenging to create APR tools and approaches that seamlessly fit into developers' workflows [41].

Recent studies have explored human factors pertaining to APR use. Most relevant to our study is research conducted by Winter and colleagues [40], who conducted a survey of 386 software developers about their bug finding and fixing practices and experiences. They concluded that developers have a strong feeling of satisfaction, trust and benefit to fix bugs manually rather than relying on APR tools, which serves as a deterrent for them from using APR tools. More precisely, this prior work suggests that developers may opt for APR tools that provide a selection of patches rather than tools that automatically apply fixes. Additionally, they found that developers prefer to use APR during testing, followed by during implementation, and that developers preferred APR tools that detect flaws during development. While this study sheds a light on developers' preferences regarding their use of APR tools in different stages of the software engineering process, it does not concentrate on the drawbacks that developers would like to see eliminated or the functionality they would like to see incorporated into APR tools to increase use. Additionally, there is little effort made to investigate preferred APR tools and support among developers and existing APR practices in teams or workplaces.

Additionally, Noller et al. [28] examined existing test-suite based APR tools and obtained feedback from 100 software practitioners in 2021. They revealed that the inability of current APR tools to deliver high quality patches within a top-10 ranking and an acceptable time frame of one hour was evident from both approaches, which significantly restricts software practitioners' use of APR tools. They also discovered that the level of experience has minimal bearing on developers' innate attitudes regarding APR, which are shared by the majority of developers. Wang et al. [38] sought to determine whether there was a distinction between fixes created manually by developers and those created using APR approaches then in use. They discovered that APR approaches are likely to produce easier patches compared to the developer given ones for complicated bugs. The 177 patches they worked with produced logical equivalence but syntactic disparity.

Aurora et al. [29] tried to investigate whether human reviewers accept security patches suggested by APR tools and if knowing the fact that a patch was generated by a specialized tool other than human affect their judgement. Here, in the first phase, using a balanced design, human reviewers were presented with a combination of patches proposed by APR tools for different vulnerabilities and asked to adopt or reject the proposed patches. In the second phase, participants were told that some of the proposed patches were generated by security specialized tools (even if the tool was actually a 'normal' APR tool) and measure whether the human reviewers would change their decision to adopt or reject a patch. However, the experiment was conducted in an academic setting, and to maintain power, it focused on a limited sample of popular APR tools and popular vulnerability types. Also, no detailed result section or analysis was found. The willingness of human practitioners to rely

on automation or APR tools are largely affected by trust. Hoff et. al. [11] completed a thorough analysis of empirical studies on automation trust and offered design suggestions for reliable automation. Their three-layered trust model offers an original viewpoint for understanding how trust in automation varies.

Building on insights and efforts from prior research studies, our work investigates software practitioners experiences with and perceptions of APR and APR tools. Unlike prior work, our research engages with software practitioners to conduct exhaustive evaluations that can provide detailed insights into who may be using or not using APR tools, why, and what we can do to increase the potential for widespread use.

### 3 METHODOLOGY

To gain insight into how APR tools are currently being used and address the gap between research and use in practice, our study aims to answer the following research questions:

**RQ1** What factors influence the awareness and adoption or use of APR in practice?

**RQ2** To what extent are APR tool(s) being used in practice compared to other forms of support?

**RQ3** What are the human-centric challenges faced by developers that can prevent the widespread use of APR tools?

#### 3.1 Survey Design

To answer our research questions, we designed an online survey with questions regarding experiences and requests for feedback on automated program repair technology from practitioners involved in software engineering tasks.<sup>1</sup> To design our survey, we first searched well-known social media platforms that are frequently accessed by software practitioners, such as GitHub, Reddit, Hacker News, and Stack Overflow, to acquire initial insights into APR use in practice. We gathered information on specific tools that developers are currently using as well as various blogs or posts (e.g., HackerNews posts<sup>2</sup>) to curate recent perspectives of practitioners about APR tools. Additionally, we examined prior studies that investigated software engineers' attitudes towards APR before finalizing our survey questions and options for answers that we provided.

Our final survey included four sections. First, we collected participant *demographics*. The majority of the questions in this section were influenced by the 2022 StackOverflow Developer Survey.<sup>3</sup> This includes questions regarding the respondents' personal, social, and professional background as well as questions regarding their programming language use and preferences. We then asked respondents about their experiences with APR tools within their teams and workplaces. Next, for those who had experience using APR tools, we asked questions about specific tools and frequency of use. Lastly, we asked respondents to provide their thoughts and recommendations on existing APR tools and techniques. This includes the pros and cons of current tools, alternatives they are using, and ideas for improvement.

<sup>1</sup>This research is approved under IRBNet Number 2006552-1

<sup>2</sup><https://news.ycombinator.com/item?id=19316228>

<sup>3</sup><https://survey.stackoverflow.co/2022/>

**Attention Question:** Due to the length of the survey, we added an *Attention Question* to assess whether the participant was paying enough attention to complete it. Our attention question asked respondents to select “Occasionally” among the options provided.

### 3.2 Respondents

We shared the survey with the research team’s contacts and on their social media accounts, including Twitter and LinkedIn. Respondents had the option of participating in a drawing for an Amazon gift card after completing the survey. Our efforts yielded a total of 800 responses. Among those, 732 (92%) agreed to the conditions given in our consent form and filled out the main survey. We removed invalid responses (see Section 3.3) and considered the remaining 331 valid responses for our analysis.

**Table 1: Gender of respondents**

Gender	Count	Percentage
Male	196	59.21%
Female	134	40.48%
Non-binary/Non-conforming	1	0.30%

**Table 2: Age of respondents**

Age Group	Count	Percentage
8-24	21	6.36%
25-29	82	24.85%
30-34	121	36.67%
35-39	61	18.48%
40-44	36	10.91%
45-49	9	2.73%
>=50	1	0.30%

As we have mentioned before, we asked respondents to report their age, gender and ethnicity to explore correlations with with APR awareness and experiences. The majority of our respondents identified as male or female, though we did have one valid respondent who identified as non-binary, genderqueer, gender non-conforming (Table 1). Respondents’ ages ranged from 30 to over 50 years of age (Table 2). We covered a wide range of respondents in terms of race/ethnicity including Hispanic or Latino/a, indigenous Australian or native American or pacific islander (Table 3). Among our respondents, many reported having at least one physical or mental disability (Table 5). We also saw diversity in the size of the respondents’ places of work, ranging from freelancers or sole proprietors to companies with over 10,000 employees (Table 4). Respondents also reported different levels of overall (Table 6) and professional (Table 7) coding experiences ranging from a minimum of less than a year to a maximum of more than 50 years.

### 3.3 Data Preparation

As we collected the responses for our survey online, it heightens potential for invalid responses according to prior work [3]. Therefore, before analyzing our survey responses, we conducted a series

**Table 3: Race/Ethnicity of Respondents**

Race/Ethnicity	Count	Percentage
White	254	76.74%
European	22	6.65%
Asian	15	4.53%
Indian	10	3.02%
Hispanic or Latino/a	9	2.72%
Black	9	2.72%
North American	10	3.02%
South Asian	9	2.72%
South American	3	0.91%
African	4	1.21%
Middle Eastern	2	0.60%
Southeast Asian	1	0.30%
East Asian	1	0.30%
Central American	1	0.30%
Indigenous (e.g., Native American or Indigenous Australian)	1	0.30%
Biracial	1	0.30%
Pacific Islander	1	0.30%

**Table 4: Respondents’ workplace size**

Company Size	Count	Percentage
freelancer, sole proprietor	5	1.51%
2-9 employees	10	3.02%
10-19 employees	44	13.29%
20-99 employees	135	40.79%
100-499 employees	80	24.17%
500-999 employees	45	13.60%
1000-4999 employees	10	3.02%
5000-9999 employees	0	0.00%
>=10000 employees	2	0.60%

of data cleaning and preparation steps to ensure data integrity. We excluded responses if they:

- declined to give their consent to fill out the survey form [68 responses],
- We found two unusual responses where one respondent selected all 9 physical and mental disability options and another respondent selected 7 disability options including blind, deaf, inability to walk. Therefore, we excluded these responses [2 responses],
- reported being members of all racial/ethnic groups [1 response],
- Our survey has a total of 32 questions (26 questions for practitioners with no prior experience in using any APR tool). Previous studies have shown that a decreased accuracy rate is seen among respondents who react more quickly than 10 seconds per question [15], which proposes our valid response time as a minimum of 4 minutes. However, before conducting the main survey, we gained responses from our lab mates to get an idea of the average time needed to fill out

**Table 5: Physical or mental disabilities faced by respondents**

Physical/Mental Disability	Responses	Percentage
Anxiety disorder	89	26.89%
Other mood or emotional disorders (e.g., depression, bipolar disorder, and postpartum depression )	82	24.77%
Inability to walk or climb stairs without assistance	4	1.21%
Concentration or memory disorder (e.g., ADHD)	15	4.53%
Autism or Autism Spectrum Disorder (ASD) (e.g., Asperger's)	15	4.53%
Learning differences (e.g., Dyslexia)	12	3.63%
Inability or difficulty in typing	7	2.11%
Blind or difficulty in seeing	3	0.91%
Deaf or difficulty in hearing	5	1.51%
None	187	56.50%

**Table 6: Overall Coding Experience of Respondents**

Experience (years)	Count	Percentage
<1 year	3	0.91%
1-4 years	36	10.88%
5-9 years	132	39.88%
10-14 years	73	22.05%
15-19 years	49	14.80%
20-24 years	20	6.04%
25-29 years	13	3.93%
30-34 years	4	1.21%
35-39 years	1	0.30%
40-44 years	0	0.00%
45-49 years	0	0.00%
>=50 years	0	0.00%

**Table 7: Professional Experience of Respondents**

Experience (years)	Responses	Percentage
<1 year	17	5.14%
1-4 years	128	38.67%
5-9 years	82	24.77%
10-14 years	53	16.01%
15-19 years	36	10.88%
20-24 years	8	2.42%
25-29 years	5	1.51%
30-34 years	1	0.30%
35-39 years	0	0.00%
40-44 years	0	0.00%
45-49 years	0	0.00%
>=50 years	1	0.30%

the survey form properly. We then agreed to the fact that if someone attentively fills out the form, s/he will need at least 3minutes considering the length of the survey, as well as, the quantity and type of open-ended questions we had. Therefore, we deleted responses of those who completed the survey in under 3 minutes [125 responses],

- provided responses that did not clearly or accurately map to the question (e.g., answering "email" as an APR tool they have used) [72 responses],
- provided duplicate responses [7 responses],
- answered the attention question incorrectly [289 responses],
- selected 'No' for knowledge of APR tools but 'Yes' to experience using APR tools [11 responses], and
- provided responses to open ended questions generated by ChatGPT (e.g., following patterns like <Keyword> : <Definition> [5 responses]

We considered the responses of the remaining 331 participants as valid and analyzed the valid responses only.

**3.3.1 Categorization of valid data.** We used two questions to determine participants' knowledge and experience with APR: "Q21: Do you know what Automated Program Repair is?" and "Q24: Have you ever used any Automated Program Repair tool?". We combined the responses for these two questions to categorize the respondents into three separate categories based on their APR awareness:

- Level 0 (APR-Unaware):** Those who selected "No" for both the questions (22 respondents), that means those who did not have any knowledge or prior experience using APR tool, were considered as APR-Unaware.
- Level 1 (APR-Aware):** Those who selected "Yes" for Q21 but "No" for Q24 (34 respondents), that means those who have knowledge about APR but have never used any APR tool were considered as APR-Aware.
- Level 2 (APR-User):** Those who selected "Yes" for both the questions (275 respondents), that means those who had knowledge and prior experience using APR tool(s), were considered as APR-User.

To understand the particular effects of job position, we divided our 14 job positions into three separate categories:

- Managers:** Those who selected any job positions related to Management (Project Manager, Engineering Manager, Product Manager and Senior Executive) were labeled as Managers (100 respondents).



- *QA Specialists*: If someone was not labeled yet and selected any option related to QA (Developer, QA or test; QA, Automation Engineer and QA Specialist) were labeled as QA Specialists (47 respondents).
- *Developers*: All other respondents were labeled as Developers (179 respondents).

In order to more clearly grasp the specific effects of company size on knowledge or experience in APR tool(s), we categorized company size into three different groups:

- small companies: companies with employees less than 100 (192 respondents).
- medium companies: companies with 100-999 employees (125 respondents).
- large companies: companies with 1000 or more than 1000 employees (12 respondents).

To better understand the dependency or the effect of both professional and total experience in knowledge and experience in APR tool(s), we divided experience into three separate categories:

- *Low experienced*: respondents with experience less than 5 years (145 respondents).
- *Medium experienced*: respondents with experience from 5 to 14 years (135 respondents).
- *Highly experienced*: respondents with experience of 15 or more than 15 years (51 respondents).

Similarly, we divided respondents into three age groups:

- Age Group 1: respondents from 18 to 29 years old (103 respondents).
- Age Group 2: respondents from 30 to 39 years old (182 respondents).
- Age Group 3: respondents of 40 or 40+ years old (46 respondents).

### 3.4 Data Analysis

To answer RQ1, we performed Chi-Square tests to establish a relationship of the factors (job position, company size, total and professional experience, race/ethnicity, age group, language used or preferred) with knowledge of APR and experience in using APR tool(s). To get better ideas about these relationships, we combined our 14 job positions into three separate categories and categorized each respondent based on their APR awareness level. Similarly, we divided the workplace size, experience (total and professional) and age group of respondents into three different subgroups separately. Then we performed Kruskal-Wallis test followed by Post-Hoc Mann Whitney U test to determine which groups were significantly different. For significantly different pairs, we calculated the mean to infer the direction. For answering both RQ2 and RQ3, we used descriptive analysis. For RQ2, we analyzed the responses to compare the use of existing APR tool(s) to other available or common support(s) for finding or fixing bugs. Similarly, we analyzed the perspectives or objections of our respondents about the current APR tool(s) to address the human-centric challenges faced by software practitioners.

While most of our survey questions were multiple choice, we included two open ended questions at the end of the survey to solicit additional insights practitioners may want to share their preferences regarding necessary and unnecessary APR tool support.

Given we only got a small number of responses to these questions (35 total), took a thematic summarization approach to analysis. After extracting the responses provided, we grouped responses for each question by high level theme (e.g., automation) and summarized the sentiments shared within each of our themes (e.g., issues with end-to-end automation).

## 4 FINDINGS

In the following sections, we discuss the findings from our survey. We collected responses to a standard set of demographic questions following our survey to better understand whether our sample represents the general population of the software development community. The factors - disability, gender, and race were not the focus of this study, and therefore we do not present any hypothesis or tests based on these demographics. We feel this information is critical to collect as it may enable us to conduct future, more focused, studies exploring, for example, how APR tools support (and can better support) developers with disabilities. Our analyses provided insights into factors that correlate with APR awareness and use, practitioner engagement with APR, and the human-centric barriers and challenges that may be impeding widespread adoption and use.

### 4.1 Factors Influencing APR in Practice (RQ1)

We considered several factors (job position, company size, experience, race/ethnicity, age group, preferred or used language) to answer RQ1. We performed chi-square tests to examine which factors influence knowledge or usage of APR tools in practice. Based on our data, we will discuss each factor and its impact.

**4.1.1 Job Position.** We conducted a chi-square test of independence to examine the relationship between job position and APR awareness. We found a significant relationship ( $p = 0.035$ ) between knowledge of APR and job position. Similarly, we compared job position and APR experience. Our chi-square test failed to identify a significant relationship between these variables. We performed a Kruskal-Wallis test to determine whether there is significant difference in APR awareness [as defined in Section 3.3.1] between the three job positions types (Managers, QA specialists and Developers). The Kruskal-Wallis H test indicated that there is a significant difference in the dependent variable (job position) between the different groups ( $X^2(2) = 7.54, p = 0.023$ ). As shown in Table 8, a Post-Hoc Mann Whitney U test using a Bonferroni corrected alpha of 0.017 indicated that the mean rank of developers and managers ( $x_{dev} - x_{man}$ ) is significantly different. We calculated the means for both developers ( $\mu = 1.83$ ) and managers ( $\mu = 1.61$ ) to infer the direction. Our findings suggest that developers are more aware of APR than managers.

**Table 8: Post-Hoc Mann Whitney U Test for APR Awareness based on Job Role Factor**

Pair	H statistic	Critical value	p-value
$x_{dev} - x_{man}$	5.9124	5.7308	<b>0.01503</b>
$x_{dev} - x_{qa}$	0.3308	5.7308	<b>0.5652</b>
$x_{man} - x_{qa}$	3.9224	5.7308	<b>0.04765</b>

**4.1.2 Company size.** We used a chi-square test to determine whether there was a correlation between company size and awareness of APR, and found a significant relationship ( $p = 1.048e^{-8}$ ). We also found a significant correlation between company size and prior experience using APR tools ( $p = 0.00006$ ). We then performed a Kruskal-Wallis test to determine whether there is any significant distinction between APR awareness level [Section 3.3.1] based on company size. The Kruskal-Wallis H test indicated that there is a significant difference in the dependent variable (company size) between the different groups,  $X^2(2) = 11.58$ ,  $p = .003$ . As shown in Table 9, a Post-Hoc Mann Whitney U test using a Bonferroni corrected alpha of 0.017 indicated that the mean ranks of small and medium ( $x_{small} - x_{med}$ ) as well as small and large ( $x_{small} - x_{large}$ ) companies are significantly different. Given these significant differences, we calculated the means for small companies ( $\mu = 1.86$ ), medium companies ( $\mu = 1.66$ ) and large companies ( $\mu = 1.22$ ) to infer the direction and found that employees of small companies are more aware of APR than employees of both medium or large companies.

**Table 9: Post-Hoc Mann Whitney U Test for APR Awareness based on Company Size Factor**

Pair	H statistic	Critical value	p-value
$x_{small} - x_{med}$	5.7961	5.7308	0.01606
$x_{small} - x_{large}$	8.9154	5.7308	0.002828
$x_{med} - x_{large}$	2.7873	5.7308	0.09501

**4.1.3 Coding Experience.** To evaluate correlation between coding experience and knowledge or experience of APR, we considered overall coding experience and professional coding experience (excluding academic) separately. For awareness of APR [Section 3.3.1] and overall coding experience, we failed to establish any significant relationship ( $p = 0.146$ ). However, for professional coding experience, the chi-square test revealed a significant correlation ( $p = 0.00001$ ). For experience using APR tools, for both overall and professional coding experience, the relationships were significant ( $p = 0.0018$  for overall and  $p = 0.014$  for professional). The Kruskal-Wallis H test indicated a significant difference in the dependent variable (overall coding experience, including academic coding) between low, medium and highly experienced respondents ( $X^2(2) = 17.02$ ,  $p < .001$ ). As shown in Table 10, a Post-Hoc Mann Whitney U test using a Bonferroni corrected alpha of 0.017 indicated that the mean ranks of low and medium experience ( $x_{low} - x_{med}$ ) as well as low and high experience ( $x_{low} - x_{high}$ ) are significantly different.

Given these findings, we calculated the means for all low ( $\mu = 1.42$ ), medium ( $\mu = 1.81$ ) and high ( $\mu = 1.75$ ) experience respondents. Our analyses indicated that medium and high experienced respondents are more aware of APR than respondents with low experience. The Kruskal-Wallis H test failed to indicate a significant difference in the dependent variable (professional experience, excluding academic coding experience) between the different groups,  $X^2(2) = 1.6$ ,  $p = .434$ .

**Table 10: Post-Hoc Mann Whitney U Test for APR Awareness based on Total Coding Experience Factor**

Pair	H statistic	Critical value	p-value
$x_{low} - x_{med}$	16.7401	5.7308	0.00004287
$x_{low} - x_{high}$	8.6008	5.7308	0.00336
$x_{med} - x_{high}$	0.3091	5.7308	0.5782

**Table 11: Post-Hoc Mann Whitney U Test for APR Awareness based on Preferred Language**

Pair	H statistic	Critical value	p-value
$x_{java} - x_{python}$	11.2924	6.9605	0.0007783
$x_{java} - x_{c++}$	1.3246	6.9605	0.2498
$x_{java} - x_{js}$	21.2177	6.9605	0.0000041
$x_{python} - x_{c++}$	3.413	6.9605	0.06468
$x_{python} - x_{js}$	1.724	6.9605	0.1892
$x_{c++} - x_{js}$	8.7331	6.9605	0.003125

**4.1.4 Age.** We performed chi-square test for age and knowledge of APR and found significant relationship ( $p = 0.0004$ ). For age and experience using APR tools, we also established significant relationship ( $p = 0.017$ ) by performing chi-square test. However, a Kruskal-Wallis H test failed to establish any significant differences between pairs of age groups ( $X^2(2) = 5.69$ ,  $p = .058$ ).

**4.1.5 Programming Language.** To investigate relationships with programming languages, we considered most used language and preferred language among respondents separately. We performed chi-square tests for knowledge of APR (Section 3.3.1) and found both most used language and preferred language were significant factors ( $p = 0.044$  and  $p = 0.332$ , respectively). Similarly, for experience using APR tools, we performed chi-square tests and found significant correlations for both most used language and preferred language ( $p = 0.02$  and  $p = 0.00003$ , respectively). Given these findings, we performed a Kruskal-Wallis test for the four most used languages among respondents (Java, Python, C++, Javascript). However, this test failed to identify significant differences between pairs of these languages,  $X^2(2) = 4.33$ ,  $p = .228$ .

Additionally, we performed a Kruskal-Wallis test for the four most preferred languages among respondents (Java, Python, C++, Javascript). The test indicated that there is a significant difference in the dependent variable (preferred language) between the different groups ( $X^2(3) = 24.2$ ,  $p < .001$ ). According to a Post-Hoc Mann Whitney U test using a Bonferroni corrected alpha of 0.0083 in Table 11, the mean ranks of Java and Python ( $x_{java} - x_{python}$ ), Java and Javascript ( $x_{java} - x_{js}$ ), as well as C++ and Javascript ( $x_{c++} - x_{js}$ ) are significantly different. We calculated means for respondents preferring Java ( $\mu = 1.91$ ), Python ( $\mu = 1.67$ ), C++ ( $\mu = 1.85$ ) and Javascript ( $\mu = 1.54$ ). Our findings suggest practitioners who prefer Java are more aware of APR than those who prefer Python or Javascript and that those preferring C++ are more aware of APR than those preferring Javascript.

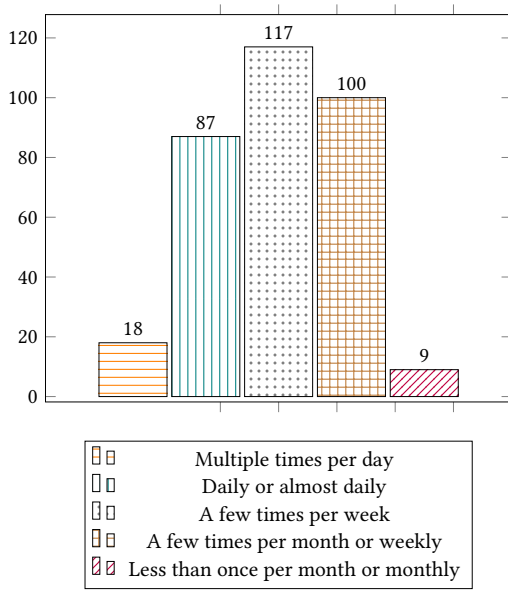


Figure 1: Frequency of fixing code or bugs

We found significant relationships between APR awareness and several factors, including: job role; company size; overall coding experience; and preferred programming language. On the whole, developers are more aware of APR than managers. We also found that employees of small companies (<100 employees) are more likely to be aware of APR than employees of medium or large companies. Practitioners with more of overall coding experience (>5 years) are more likely to be aware of APR. Lastly, developers who prefer Java over Python and Javascript and C++ over Javascript are more likely to be aware of APR.

#### 4.2 APR Tool Use vs. Other Support (RQ2)

We asked respondents about how often they fix bugs in their code. As shown in Figure 1, most of our respondents (117) reported fixing bugs in their code a few times per week. Most others reported either having to fix their code a few times a month/weekly (100) or having to fix bugs daily/almost daily (87). Very few responded that their frequency of fixing bugs as multiple times per day (18 responses) or less than once per month or monthly (9 responses). As with prior work, this emphasizes the time commitment involved in fixing software defects [16, 33].

Figure 2 provides a visual summary of the frequency with which our respondents are using APR tools for fixing bugs. Of our 331 respondents, 42.7% reported utilizing automated program repair tools on a weekly or monthly basis. For 31.6% of our respondents, they use APR tools less frequently (a few occasions per week). Along with data on APR tool use, we also collected data on other forms of support practitioners may be using. Table 12 shows the summary of the responses from our survey, where respondents could select *multiple times per day* (5), *daily or almost daily* (4), *a few times per week* (3), *a few times per month or weekly* (2), and *less than once per month or monthly* (1). As we can see, our respondents use other forms of support for fixing bugs in their code more often

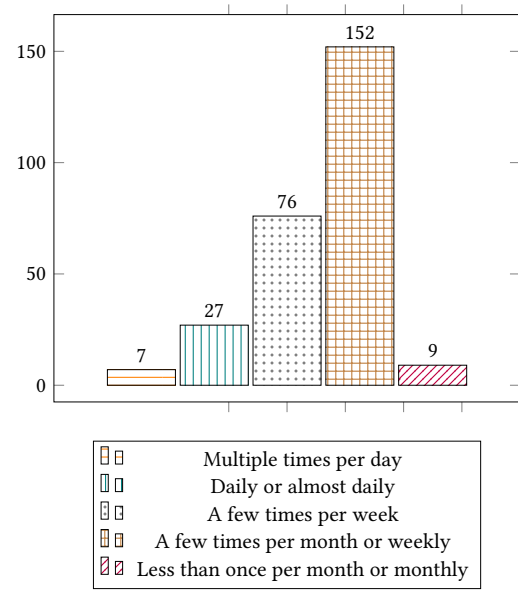


Figure 2: Frequency of using APR tool(s)

Table 12: Frequency of the usage of existing APR tool(s) compared to other forms of support from 1 (less than once per month or monthly) to 5 (multiple times per day)

Forms of support	Frequency of use					Spark line	Mean Score
	5	4	3	2	1		
APR tool(s)	7	27	76	152	10		2.52
MDN	11	45	157	95	23		2.78
Reddit	19	45	137	106	24		2.79
Medium	11	51	167	79	23		2.84
Talk to someone from other industry	22	96	88	86	39		2.93
ChatGPT	21	64	148	81	17		2.97
Stack Overflow	21	66	134	75	15		3.01
GitHub	18	71	156	71	15		3.02
Talk to someone from other team	27	91	110	84	19		3.07
Talk to someone from own team	34	118	88	81	10		3.26

compared to using APR tool(s). Most prefer to talk to someone from their own team or other team to fix the issues. Excluding the human support, among other available supports they prefer GitHub, Stack Overflow and ChatGPT over using APR tools.

We asked respondents who have used automated program repair tools previously additional questions to get an idea of how familiar or experienced they are in using these tools. They were also asked about the details of the tool(s) they use. For existing tools, respondents mentioned a number of tools such as Adrenaline,<sup>4</sup> Eclipse's

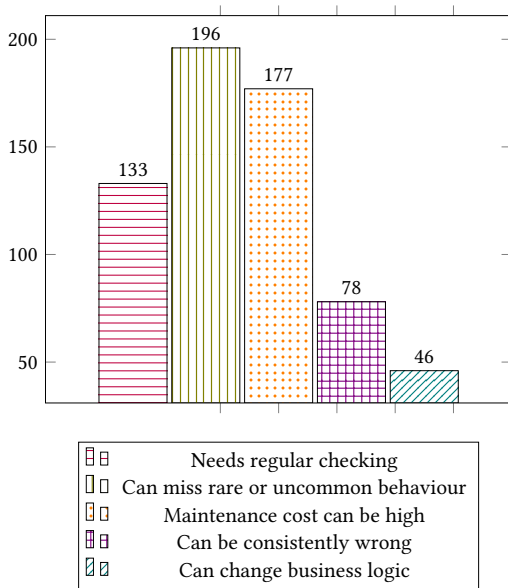
<sup>4</sup><https://useadrenaline.com/>

Quick Fix,<sup>5</sup> Test Disk,<sup>6</sup> KLEE,<sup>7</sup> and Nopol [44]. The most common name mentioned by 8 respondents (1%) was ASTOR [21]. This low response rate was caused by the fact that this question was open-ended. Only 26 out of 331 respondents gave valid answers for this question (We excluded responses “N/A, N, no, Not Applicable, A” etc. considering invalid responses).

*Most respondents reported a preference for human support from own team or other team of their own company for fixing any issue or bug over using automated program repair. Though respondents report using APR tools on a frequent basis, they also report a preference for other forms of support, such as GitHub, Stack Overflow and ChatGPT.*

### 4.3 Human-centric Barriers (RQ3)

Our survey also included questions regarding perceptions of the drawbacks of automated program repair. We integrated existing disadvantages mentioned in online programming communities and found in prior work [40] to investigate barriers to APR adoption and use. We also collected insights from respondents on the effects of these drawbacks and improvements they wish to see in existing APR tools and techniques.



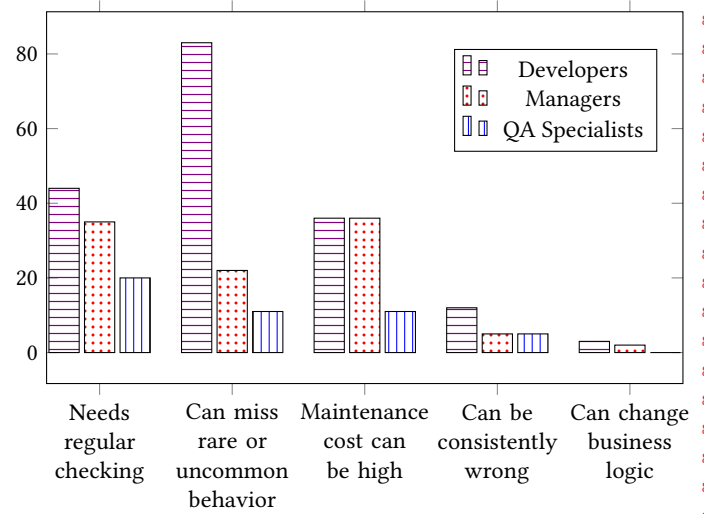
**Figure 3: Disadvantages faced by software practitioners**

Figure 3 summarizes the distribution of disadvantages reported by our respondents. We provided options of some common existing disadvantages for this question. Existing common disadvantages were collected from common platforms for software practitioners like GitHub, Reddit and Stack Overflow and existing research paper [40] which were addressed by the provided options of our survey.

<sup>5</sup><https://help.eclipse.org/latest/index.jsp?topic=%2Forg.eclipse.jdt.doc.user%2Frefere%2Fref-java-editor-quickfix.htm>

<sup>6</sup><https://www.cgsecurity.org/wiki/TestDisk>

<sup>7</sup><http://klee.github.io/>



**Figure 4: Effect of job position on the perspective of disadvantage(s)**






We attempted to obtain a detailed perspective from the practitioners regarding the drawbacks of the current APR tools so that we can decide which drawbacks we should concentrate on addressing. The biggest disadvantages cited was that APR tools may miss rare or uncommon behavior (196 responses) and that the maintenance cost of APR tool(s) can be high (177 responses). Another common concern of some practitioners (133 responses) was that their contributions would need regular checking. Two other concerns reported were that APR tool(s) can be consistently wrong (78 responses) and they can change business logic (46 responses).

According to our respondents, these disadvantages effect the use or adoption of automated program repair to varying degrees. Table 13 shows the summary of the responses from our survey, where respondents could select *very affected* (5), *somewhat affected* (4), *neither affected nor unaffected* (3), *somewhat unaffected* (2), and *unaffected* (1). Respondents reported being mostly affected by the potential for missing rare or uncommon behavior, followed by maintenance costs and consistently providing wrong fixes. We also asked respondents to report on the effects fixing these disadvantages would have on the likelihood that they would use APR tools. Table 14 summarizes our survey responses, where respondents could select *very likely* (5), *somewhat likely* (4), *neither likely nor unlikely* (3), *somewhat unlikely* (2), and *very unlikely* (1). Our findings indicate generally neutral feelings about changes in their use of APR tools with reduction of the reported disadvantages.

Following these insights, we also wanted to determine if job role has any effect on practitioners' perspectives regarding the disadvantages of existing APR tools. As shown in Figure 4, there are impacts on the perspective about the disadvantages of APR tool(s) based on respondents' job role. More specifically, our findings indicate that while for most disadvantages there are similarities in perceptions across job roles, developer respondents find missing rare or uncommon behavior to be an issue much more than managers or QA specialists. On the other hand, managers are more concerned about








**Table 13: Effects of disadvantage(s) on software practitioners from 1 (unaffected) to 5 (very affected)**

Disadvantage(s)	Effect of disadvantage(s)					Spark line	Mean Score
	1	2	3	4	5		
Needs regular checking	19	59	152	91	10		3.04
Can be consistently wrong	11	93	103	94	30		3.12
Can miss rare or uncommon behavior	12	74	95	121	29		3.24
Maintenance cost can be high	14	49	138	104	26		3.24
Can change business logic	6	54	140	95	36		3.31

regular checking (35 out of 100 managers) and high maintenance cost issue (36 out of 100 managers), whereas QA specialists are concerned about regular checking (20 out of 47 QA specialists), missing rare or uncommon behavior (11 out of 47 QA specialists) and high maintenance cost issue (11 out of 47 QA specialists).

**Table 14: Probability of software practitioners' usage of existing APR tool(s) if mentioned disadvantage(s) are fixed from 1 (unlikely) to 5 (very likely)**

Disadvantage(s)	Likelihood of using APR tool(s) if the disadvantage is fixed					Spark line	Mean Score
	1	2	3	4	5		
Can miss rare or uncommon behavior	10	61	138	91	31		3.22
Can change business logic	7	50	133	103	38		3.35
Needs regular checking	7	45	132	113	34		3.37
Maintenance cost can be high	7	33	154	100	37		3.38
Can be consistently wrong	8	41	133	112	37		3.39

**Reducing barriers to use.** To better understand barriers to widespread use of APR and how to reduce them, we asked respondents to provide suggestion for APR tool features that they feel should be added or removed. Most of the suggestions for improvement provided by respondents focused on features that should be added or improved in existing APR tools and techniques. Some respondents re-iterated improvements based on disadvantages listed in the survey, such as “needs to maintain business logic”. For other respondents, the improvements needed are broader and go beyond the set of disadvantages already outlined. Based on our responses, practitioners may be more likely to use APR tools that “quickly suggest code fixes” that “consider underlying issues” and can be integrated “based on developer insight.” One way to make such an improvement suggested by one of our respondents is to provide multiple fix options for a given defect, which further validates insights from prior work on automated bug fixes in practice [4].

As indicated in our prior responses, practitioners worry about incorrect fixes or unintended problems. So APR tools should also

“include warnings or alerts of potential issues” to support applying the best, most appropriate fixes. We also found recommendations to support the ability for APR tools to learn from previous repairs to “improve the quality and accuracy of future repairs.”

A few respondents made suggestions for features that need to be removed from existing APR tool(s). Many of these suggestions are the alternative of the above suggested improvements. For example, one of the main issues brought up by respondents is the focus on end to end automation when providing fixes [38]. Respondents took issues both with “automatic patching of existing code”, particularly without final approval from the user, and “automatically generating code from scratch.” While there are cited benefits to automation in software development [23, 31, 35], our respondents noted that automation in this context can lead to “code that is difficult to maintain and debug” and even “potentially dangerous changes.” A couple of respondents effectively summarized the insights from our study when it comes to improving APR tools, noting the need to reduce “overly complex or impractical features that can increase the complexity and difficulty of using the tool” and provide “an intuitive user interface and workflow, to allow users to easily understand and manage the repair process.”

Respondents were most concerned about missing any rare or uncommon behavior; this was more-so the case for developers compared to managers and QA specialists. QA specialists and managers are both concerned about regular checking and high maintenance cost the most, where QA specialists have concern for missing rare or uncommon behavior too. Practitioners prefer APR tools that provide suggestions for multiple possible solutions rather than automatically fixing defects. Practitioners also want intuitive interfaces that provide warning or alerts for all possible changes and provide support throughout the repair process.

## 5 THREATS TO VALIDITY

We designed and administered our survey with the goal of collecting data from diverse software practitioners who are or could be potential APR users. Below we discuss some potential threats to the validity of our findings.

**External Validity.** Our survey advertisement was limited to software practitioners we could reach through our networks. Being our networks are mostly US-based, English-speaking, institutions and organizations, this limitation to location may influence the ability to generalize our insights to global software development. Also, survey respondents completed the survey on a voluntary basis, which can also have an effect on the validity of our findings. We focused on software practitioners and their job position or workplace size and did not consider workplace culture, as well as, the industry type. While we were not able to exhaustively cover the spectrum of software developers in our sample, we made intentional efforts when advertising our survey to acquire as much diversity as possible.

**Internal Validity.** We tried to choose appropriate statistical tests for the best result based on our questions or response types. We

used non-parametric test, however given final sample size was relatively small (331 responses), there may be additional significant correlations or relationships we did not discover in our analysis.

**Construct Validity.** We relied on self-reporting, such as the questions related to frequency of fixing code/bugs or using APR tool(s), which may not be as accurate as direct observation. However, this choice affords us the capability to gather data from comparatively broader set of practitioners. Survey responses are also prone to several types of bias. While it is impossible to ensure 100% removal of all invalid responses, to mitigate this threat we only included responses that survived a series of data cleaning steps (Section 3.3).

## 6 DISCUSSION

### 6.1 Increasing Use of APR in Practice

Our findings, along with findings from various prior studies [16, 17, 33, 40], indicate that a large amount of practitioner time and effort is being put into fixing software defects (see Figure 2). While previous studies suggest that APR can lighten practitioners' workload and support quickly repairing software defects [18, 40], our study found that existing APR tools are used infrequently and that many practitioners are not aware of the APR tools and techniques they could be using. While our findings suggest practitioners may not be using APR tools frequently, our findings also provide insights into ways in which we can improve APR tools to increase the potential for widespread use. We found that the forms of support practitioners are most often using involve human, or human-like, interactions (see Table 12). Furthermore, respondents noted issues with the fully automated application of code modifications to fix defects (Section 4.3). This is perhaps because for practitioners there is a certain level of trust required when adopting and regularly using software tools [14].

Therefore, one way we can increase the use of APR tools in practice, is by working towards APR tools that mimic the interactions developers have with other developers, which our findings suggests may be one of the most frequently used methods for repairing software defects. Given we also found that ChatGPT is a commonly used resource, one concrete direction to explore is if and to what extent we can use ChatGPT and other large language models that attempt to provide human-like interactions during the "automated" repair process. For example, rather than providing a one shot automated fix or options for the developer to choose from, we can imagine a sort of "conversational APR" tool [42] [37] that can provide automated support for finding potential solutions, but engaging *with* the developer in the process of understanding, applying [6], and validating the best repair. This supports the ability for practitioners to be involved in the defect repair process with less overhead and potential for technical debt.

### 6.2 Increasing Awareness of APR Tools

Despite the years of research on automated program repair [10, 19] and the widespread availability of APR tools and techniques [40], our study found that one reason APR tools are not more widely used may be due to lack of awareness regarding their availability. Our findings also indicate a diversity of factors that correlate with more or less awareness of APR tools and techniques (Section 4.1).

Prior research suggests one of the most effective ways to increase awareness and adoption of software tools is through practitioners' peers [14, 26]. Prior work has also found that the peer interactions that can facilitate the discovery of new tools may be infrequent in practice [26]. Findings from our study further emphasize the infrequency of these interactions, given differences in awareness based on job role (Section 4.1.1), overall coding experience (Section 4.1.3), preferred language, and company size (Section 4.1.2).

While our work, and prior studies, provide important insights into practitioner awareness and adoption of APR tools, there are still foundational gaps in our understanding of *why* awareness varies across different practitioner populations. For example, our survey found that practitioners who work in small companies may be more likely to be aware of APR and existing tools than those working in larger companies. One reason for this could be that while large companies tend to have employees dedicated to the various kinds of tasks involved in developing software, small companies have fewer employees leading to less segmented roles [7]. Furthermore, smaller companies and teams enable better communication which studies have shown can lead to increased awareness and adoption of tools [7, 14, 26]. Smaller companies also have fewer resources, which may lead to a need for seeking support in places larger companies would not (e.g., using an off-the-shelf APR tool versus in-house tools that may not include, need, or be explicitly labeled as automated program repair) [32]. In this case, we can speculate regarding the various nuances to practitioners tool seeking and finding. However, still, an important direction of future research would be to better understand the populations that are aware of APR, how they became aware of the tools they use, and how we can better facilitate the introduction of these tools and techniques to practitioners.

Our findings also indicated a relationship between practitioners' preferred language and awareness of APR. More specifically, we found that practitioners who prefer Java over Python or JavaScript are more aware of APR tools (Section 4.1.5). The primary cause of this might be that, with a 35% share, Java is the most widely used programming language for test automation [1]. Furthermore, the vast majority of the research and development that has been done to support automated program repair over the years has focused on providing support for Java programs [8, 20, 25]. However, with the increase in development using Python, we are seeing an increase in research and development on providing support for APR in Python programs [34, 43, 46]. Our findings suggest there is still much work to be done in providing and exploring cross-language APR support to realize the increased awareness experienced by Java developers.

## 7 CONCLUSION

This paper presented the perspectives of 331 software practitioners about the existing APR tool(s) and the disadvantages that they face while using the tools. We analyzed their responses and established important relationships of the practitioners' demographics with their knowledge about APR and their experience in using APR tool(s). Additionally, we discussed significant differences among respondents based on several demographics data. We focused on the human-centric disadvantages faced by the practitioners while using existing APR tools and the effect every disadvantage has on

their work. There is an opportunity to work on these disadvantages to fix or improve them for encouraging software practitioners to use APR tool(s) more effectively.

## REFERENCES

- [1] 2017. The Leading Test Automation Methods Are Revealed! <https://blog.testproje.io/2017/12/13/leading-test-automation-methods/> [2017-13-12].
- [2] 2022. Real-life consequences of software bugs. <https://www.softwarerepairingmagazine.com/knowledge/5-real-life-consequences-of-software-bugs-why-high-quality-standards-are-so-important/> [2022-28-03].
- [3] Patricia Al-Salom and Carlin J Miller. 2019. The problem with online data collection: Predicting invalid responding in undergraduate samples. *Current Psychology* 38 (2019), 1258–1264.
- [4] Titus Barik, Yoonki Song, Brittany Johnson, and Emerson Murphy-Hill. 2016. From quick fixes to slow fixes: Reimagining static analysis resolutions to enable design space exploration. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 211–221.
- [5] Tom Britton, Lisa Jeng, Graham Carver, Paul Cheak, and Tomer Katzenellenbogen. 2013. Reversible debugging software. *Judge Bus. School, Univ. Cambridge, Cambridge, UK, Tech. Rep* 229 (2013).
- [6] Rubens Copche, Yohan Duarte Pessanha, Vinicius Durelli, Marcelo Medeiros Eler, and Andre Takeshi Endo. 2023. Can a Chatbot Support Exploratory Software Testing? Preliminary Results. *arXiv preprint arXiv:2307.05807* (2023).
- [7] Michael A Cusumano. 1997. How Microsoft makes large teams work like small teams. *MIT Sloan Management Review* 39, 1 (1997), 9.
- [8] Zhiyu Fan, Xiang Gao, Martin Mirchev, Abhik Roychoudhury, and Shin Hwei Tan. 2023. Automated repair of programs from large language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1469–1481.
- [9] Luca Gazzola, Daniela Micucci, and Leonardo Mariani. 2018. Automatic software repair: A survey. In *Proceedings of the 40th International Conference on Software Engineering*. 1219–1219.
- [10] Claire Le Goues, Michael Pradel, and Abhik Roychoudhury. 2019. Automated program repair. *Commun. ACM* 62, 12 (2019), 56–65.
- [11] Kevin Anthony Hoff and Masooda Bashir. 2015. Trust in automation: Integrating empirical evidence on factors that influence trust. *Human factors* 57, 3 (2015), 407–434.
- [12] Yue Jia, Ke Mao, and Mark Harman. 2018. Finding and fixing software bugs automatically with SapFix and Sapienz.
- [13] Jiajun Jiang, Yingfei Xiong, Hongyu Zhang, Qing Gao, and Xiangqun Chen. 2018. Shaping program repair space with existing patches and similar code. In *Proceedings of the 27th ACM SIGSOFT international symposium on software testing and analysis*. 298–309.
- [14] Brittany Johnson, Christian Bird, Denae Ford, Nicole Forsgren, and Thomas Zimmermann. 2023. Make Your Tools Sparkle with Trust: The PICSE Framework for Trust in Software Tools. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 409–419.
- [15] Takumi Kato and Taro Miura. 2021. The impact of questionnaire length on the accuracy rate of online surveys. *Journal of Marketing Analytics* 9, 2 (2021), 83–98.
- [16] Amy J Ko and Brad A Myers. 2004. Designing the whyline: a debugging interface for asking questions about program behavior. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 151–158.
- [17] Pavneet Singh Kochhar, Xin Xia, and David Lo. 2019. Practitioners’ Views on Good Software Testing Practices. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 61–70. <https://doi.org/10.1109/ICSE-SEIP.2019.00015>
- [18] Guangliang Liu, Yang Lu, Ke Shi, Jingfei Chang, and Xing Wei. 2019. Mapping Bug Reports to Relevant Source Code Files Based on the Vector Space Model and Word Embedding. *IEEE Access* 7 (2019), 78870–78881. <https://doi.org/10.1109/ACCESS.2019.2922686>
- [19] Kui Liu, Li Li, Anil Koyuncu, Dongsun Kim, Zhe Liu, Jacques Klein, and Tegawendé F Bisseyandé. 2021. A critical review on the evaluation of automated program repair systems. *Journal of Systems and Software* 171 (2021), 110817.
- [20] Kui Liu, Shangwen Wang, Anil Koyuncu, Kisub Kim, Tegawendé F Bisseyandé, Dongsun Kim, Peng Wu, Jacques Klein, Xiaoguang Mao, and Yves Le Traon. 2020. On the efficiency of test suite based program repair: A systematic assessment of 16 automated repair systems for java programs. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 615–627.
- [21] Matias Martinez and Martin Monperrus. 2014. ASTOR: Evolutionary automatic software repair for Java. *arXiv preprint arXiv:1410.6651* (2014).
- [22] Sergey Mechtaev, Manh-Dung Nguyen, Yannic Noller, Lars Grunske, and Abhik Roychoudhury. 2018. Semantic program repair using a reference implementation. In *Proceedings of the 40th International Conference on Software Engineering*. 129–139.
- [23] Tim Menzies, Oussama Elrawas, Jairus Hihn, Martin Feather, Ray Madachy, and Barry Boehm. 2007. The business case for automated software engineering. In *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering*. 303–312.
- [24] Martin Monperrus. 2018. Automatic software repair: A bibliography. *ACM Computing Surveys (CSUR)* 51, 1 (2018), 1–24.
- [25] Manish Motwani, Mauricio Soto, Yuriy Brun, Rene Just, and Claire Le Goues. 2020. Quality of automated program repair on real-world defects. *IEEE Transactions on Software Engineering* 48, 2 (2020), 637–661.
- [26] Emerson Murphy-Hill and Gail C Murphy. 2011. Peer interaction effectively, yet infrequently, enables programmers to discover new tools. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work*. 405–414.
- [27] Nicholas Nethercote and Julian Seward. 2003. Valgrind: A program supervision framework. *Electronic notes in theoretical computer science* 89, 2 (2003), 44–66.
- [28] Yannic Noller, Ridwan Shariffdeen, Xiang Gao, and Abhik Roychoudhury. 2022. Trust enhancement issues in program repair. In *Proceedings of the 44th International Conference on Software Engineering*. 2228–2240.
- [29] Aurora Papotti, Ranindya Paramitha, and Fabio Massacci. 2022. On the acceptance by code reviewers of candidate security patches suggested by Automated Program Repair tools. *arXiv preprint arXiv:2209.07211* (2022).
- [30] Chris Parnin and Alessandro Orso. 2011. Are automated debugging techniques actually helping programmers?. In *Proceedings of the 2011 international symposium on software testing and analysis*. 199–209.
- [31] Chris Parnin and Alessandro Orso. 2011. Are automated debugging techniques actually helping programmers?. In *Proceedings of the 2011 international symposium on software testing and analysis*. 199–209.
- [32] Francisco J Pino, Félix García, and Mario Piattini. 2008. Software process improvement in small and medium software enterprises: a systematic review. *Software Quality Journal* 16 (2008), 237–261.
- [33] Strategic Planning. 2002. The economic impacts of inadequate infrastructure for software testing. *National Institute of Standards and Technology* 1 (2002).
- [34] Julian Aron Prenner, Hlib Babii, and Romain Robbes. 2022. Can OpenAI’s codex fix bugs? an evaluation on QuixBugs. In *Proceedings of the Third International Workshop on Automated Program Repair*. 69–75.
- [35] RM Sharma. 2014. Quantitative analysis of automation and manual testing. *International journal of engineering and innovative technology* 4, 1 (2014).
- [36] Rijnard van Tonder and Claire Le Goues. 2018. Static automated program repair for heap properties. In *Proceedings of the 40th International Conference on Software Engineering*. 151–162.
- [37] Rijnard van Tonder and Claire Le Goues. 2019. Towards s/engineer/bot: Principles for program repair bots. In *2019 IEEE/ACM 1st international workshop on bots in software engineering (BotSE)*. IEEE, 43–47.
- [38] Shangwen Wang, Ming Wen, Liqian Chen, Xin Yi, and Xiaoguang Mao. 2019. How different is it between machine-generated and developer-provided patches?: An empirical study on the correct patches generated by automated program repair techniques. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 1–12.
- [39] Elaine J Weyuker and Filippos I Vokolos. 2000. Experience with performance testing of software systems: issues, an approach, and case study. *IEEE transactions on software engineering* 26, 12 (2000), 1147–1156.
- [40] Emily Winter, David Bowes, Steve Counsell, Tracy Hall, Sæmundur Haraldsson, Vesna Nowack, and John Woodward. 2022. How do developers really feel about bug fixing? Directions for automatic program repair. *IEEE Transactions on Software Engineering* (2022).
- [41] Emily Winter, Vesna Nowack, David Bowes, Steve Counsell, Tracy Hall, Sæmundur Haraldsson, and John Woodward. 2022. Let’s talk with developers, not about developers: A review of automatic program repair research. *IEEE Transactions on Software Engineering* 49, 1 (2022), 419–436.
- [42] Emily Rowan Winter, Vesna Nowack, David Bowes, Steve Counsell, Tracy Hall, Sæmundur Haraldsson, John Woodward, Serkan Kirbas, Etienne Windels, Olayori McBello, et al. 2022. Towards developer-centered automatic program repair: findings from Bloomberg. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1578–1588.
- [43] Chunqiu Steven Xia, Xuxiang Wei, and Lingming Zhang. 2023. Automated program repair in the era of large pre-trained language models. In *Proceedings of the 45th International Conference on Software Engineering (ICSE 2023)*. Association for Computing Machinery.
- [44] Jifeng Xuan, Matias Martinez, Favio Demarco, Maxime Clement, Sebastian Lame-las Marcote, Thomas Durieux, Daniel Le Berre, and Martin Monperrus. 2016. Nopol: Automatic repair of conditional statement bugs in java programs. *IEEE Transactions on Software Engineering* 43, 1 (2016), 34–55.
- [45] Deheng Yang, Yuhua Qi, and Xiaoguang Mao. 2018. Evaluating the strategies of statement selection in automated program repair. In *Software Analysis, Testing, and Evolution: 8th International Conference, SATE 2018, Shenzhen, Guangdong, China, November 23–24, 2018, Proceedings 8*. Springer, 33–48.
- [46] Michihiro Yasunaga and Percy Liang. 2021. Break-it-fix-it: Unsupervised learning for program repair. In *International Conference on Machine Learning*. PMLR, 11941–11952.