How do developers *really* feel about bug fixing? Directions for automatic program repair

Emily Winter, David Bowes, Steve Counsell, Tracy Hall, Sæmundur Haraldsson, Vesna Nowack, John Woodward

***Abstract*—Automatic program repair (APR) is a rapidly advancing field of software engineering that aims to supplement or replace manual bug fixing with an automated tool. For APR to be successfully adopted in industry, it is vital that APR tools respond to developer needs and preferences. However, very little research has considered developers' general attitudes to APR or developers' current bug fixing practices (the activity APR aims to replace). This paper responds to this gap by reporting on a survey of 386 software developers about their bug finding and fixing practices and experiences, and their instinctive attitudes towards APR. We find that bug finding and fixing is not necessarily as onerous for developers as has often been suggested, being rated as more satisfying than developers' general work. The fact that developers derive satisfaction and benefit from bug fixing indicates that APR adoption is not as simple as APR replacing an unwanted activity. When it comes to potential APR approaches, we find a strong preference for developers being kept in the loop (for example, choosing between different fixes or validating fixes) as opposed to a fully automated process. This suggests that advances in APR should be careful to consider the agency of the developer, as well as what information is presented to developers alongside fixes. It also indicates that there are key barriers related to trust that would need to be overcome for full scale APR adoption, supported by the fact that even those developers who stated that they were positive about APR listed several caveats and concerns. We find very few statistically significant relationships between particular demographic variables (for example, developer experience, age, education) and key attitudinal variables, suggesting that developers' instinctive attitudes towards APR are little influenced by experience level but are held widely across the developer community.**

I. Introduction

AUTOMATIC program repair (APR) is a growing area of software engineering (SE) research that aims to supplement or replace manual bug fixing. In order for APR to be successfully and widely adopted, it is vital that we understand developer needs and preferences. However, we currently know little about both software developers' bug finding and fixing practices and their general attitudes towards APR. As recently as 2018, Beller et al. argued 'we have little knowledge on how software engineers debug software problems in the real world, whether they use dedicated debugging tools, and how knowledgeable they are about debugging' [1]. Similarly, Böhme highlights that 'how humans actually debug is still not really well explored' [2], stating 'given how much time practitioners spend on debugging, it [...] is a scandal how little we know about debugging'. Whilst the difficulty and frustration of bug fixing for developers is often asserted, we know even less about developers' feelings about the activity of bug fixing. Understanding developers' current bug fixing practices, and how they feel about fixing bugs, is important because the presentation of APR to developers should be based upon this understanding. As well as a lack of understanding within SE of developers' bug fixing practices (the key activity APR hopes to aid or replace), Westley Weimer's recent keynote address highlighted the lack of consideration of human factors within APR, and the need to take human factors more seriously in the discipline [3].

In our prior work [4], we conducted a literature review to assess the extent to which human factors are considered within the APR literature, as well as evaluating the quality of existing human studies. We found that there were very few human studies currently in APR – just 7% of the 260 papers we reviewed. The human studies we evaluated were of mixed quality, often involving small samples and few professional developers. In addition, we found that most APR human studies were tool specific. Fourteen of the seventeen papers with human studies that we evaluated introduced a new tool or technique that the authors had developed and then conducted a human study to test it. For example, there were several (quasi-) experimental studies evaluating how participants performed a task with or without access to the patches generated by the tool. These studies may not reflect participants' more general attitudes to APR; rather, they provide insights into the advantages and disadvantages of a proposed technique. Existing work therefore yields little insight that can be applied across APR tools and techniques or can act as guiding principles for future APR development, particularly how APR and automatically generated fixes should be presented to developers. This motivated us to perform a large-scale human study with professional developers that would investigate developers' more general, instinctive attitudes and identify the implications of this for APR development.

We argue that it is important to understand developers' current bug fixing practices, their feelings about bug fixing and their general, instinctive feelings towards APR in order to identify potential barriers to APR adoption. To this end, we designed a survey to answer the following research questions:

- RQ1: What are software developers' current bug finding and fixing practices?
- RQ2: How do software developers feel about bug finding and fixing?

E. Winter, D. Bowes, T. Hall and V. Nowack are with School of Computing and Communications, Lancaster University.

J. Woodward is with School of Electronic Engineering and Computer Science, Queen Mary University of London

S. Counsell is with Department of Computer Science, Brunel University of London

S. Haraldsson is with Department of Computing Science and Mathematics, University of Stirling

- RQ3: What are software developers' instinctive feelings towards APR?

To our knowledge this is the first paper that surveys, on a large scale, developers' general feelings towards APR. We find developers prefer APR approaches in which they maintain a role, either validating fixes or, more popularly, choosing between multiple fixes offered by a tool. This has important implications for future APR development. We also find that developers' instinctive feelings towards APR can be summarised as 'cautiously optimistic'; they are interested in and positive about the idea of APR, but they have a range of concerns, conditions and caveats.

This paper is structured as follows: Section II reports on related work; Section III describes the survey design, our sampling and recruitment strategies, and analysis process; Section IV describes the demographics of our participants. We then report our findings (Section V), and provide discussion and threats to validity in Sections VI and VII. We conclude in Section VIII.

II. Related work

There is a small body of survey-based literature that considers how software developers debug. Perscheid et al. [5], for example, conducted an online survey of 303 software developers to explore their mental models of debugging and the extent to which debugging tools developed by researchers have been adopted by professional software developers. They found that very few debugging tools were used by developers. Beller et al. [1] combined an online survey of 176 developers with observations of how developers interacted with and used a debugger in their IDE. They found that both knowledge and use of advanced debugging features was low. Beller et al. report that developers did not want more debugging features, but for existing ones to be made easier to use. These papers share a focus on *how* developers debug, rather than our own more attitudinal focus on how developers *feel* about bug finding and fixing. This is important because much of the APR literature shares an assumption that APR is replacing an inherently unsatisfying task (manual bug fixing) (see, for example, [6], [7]). We consider it important to empirically test this assumption, both to understand developers' actual attitudes towards bug fixing (as they could influence how we should present APR to developers) and to explore whether developers' feelings towards bug fixing influences their attitudes towards APR.

There have also been several studies that consider the automation of parts of the bug finding and fixing process. For example, Zou et al. [8] surveyed 337 software developers on the topic of automated bug report management techniques. Survey respondents were asked how important they considered different types of bug report management technique (e.g. bug categorisation, bug assignment). The study found that experienced developers were more negative than less experienced developers about the importance of these bug report management techniques. Similarly, Wan et al.'s survey [9] of 395 developers about defect prediction found that the most experienced respondents were the least willing to adopt defect prediction tools. The main reason for this was a lack of belief that defect prediction could work. Another similar study [10], involving a survey of 386 practitioners on the topic of fault localisation, found that 'more experienced developers perceive fault localisation to be less "essential" than less experienced ones'. This study also found a strong desire among survey participants for fault localisation techniques to provide reasoning for why parts of the program are marked as suspicious. Whilst participants were 'enthusiastic' about fault localisation research, 'they have high thresholds for adoption'. Our research similarly finds a degree of scepticism towards APR; however, we found only *very limited* evidence of any relationship between developer experience levels and their attitudes towards APR.

There have also been a small number of studies of how software developers use and interact with APR tools. Most existing human studies in APR are either controlled experiments [11] [12] [13] [14] [15] [16] or surveys [17] [18] [19]. One key feature that these APR human studies share in common is that they are tool-specific, asking participants to test, validate or give feedback on a specific APR tool or technique. As a result, they provide little insight into developers' attitudes towards APR more broadly. However, there are some exceptions. For example, Böhme et al.'s experimental study with 12 software professionals [2] found that 'practitioners are wary of debugging automation', particularly for functional bugs. Böhme et al.'s study does not paint a positive picture of software practitioners' openness to APR:

> *The majority of participants did not believe in automation due to the lack of a complete specification and due to the difficulty in code comprehension [...] for automatic program repair, participants think that it is impossible for a tool to change or add any functionality to a buggy program. Moreover, even in the presence of a complete specification, participants do not believe in automated repair due to the challenges involved in code comprehension.*

Such findings were mirrored in Parnin and Orso's experimental study with 34 developers [20], which found that 'developers were quick to disregard the tool if they felt they could not trust the results or understand how such results were computed'. Furthermore, 'the use of an automated tool helped more experienced developers find faults faster in the case of an easy debugging task, but the same developers received no benefit from the use of the tool on a harder task'. These findings are somewhat troubling for the academic SE community, as they suggest some degree of scepticism towards APR tools and techniques. However, the samples in both these studies ([2] and [20]) are small; our survey, in presenting a large sample, considers whether such perceptions towards APR are indeed commonplace.

Other related work includes a small body of literature that has considered the introduction of APR tools — at varying levels of maturity — in industry. Our own work has described the introduction of a prototype APR tool at Bloomberg [21], while there have also been studies related to Facebook's Getafix tool, for example [22]. Lessons learned

deploying Getafix include that auto-fixes should be integrated into existing development tools and predicted fast enough so as not to slow down engineers' work. Our own work at Bloomberg also demonstrated the importance of alignment with existing processes, as well as positioning the APR tool as an 'assistant' to the developer, who remains 'in the driving seat'. In this paper, we receive confirmation that developer control and involvement remains paramount, though there is less support for the idea that speed of fix generation is important for developers.

The most closely related work to our own is a survey conducted by Noller et al. of 100 software practitioners about trust in APR [23]. Noller et al.'s survey asked participants about their willingness to review automatically generated patches, how quickly they would expect results from an APR tool, participants' willingness to provide additional inputs, and the impact of providing additional information and explanations. Whilst our survey has similarities to this, we also include questions about developers' current bug finding and fixing practices. We also provide more in-depth thematic analysis than Noller et al. of the qualitative answers to open-text response questions.

As far as we aware, our survey is the first to investigate developers' current bug finding and fixing practices, their feelings about bug finding and fixing, and their general attitudes towards APR within the same survey. Our rationale for doing this is to consider any potential relationship between attitudes towards APR and feelings about bug finding and fixing, and to more fully understand the relationship between APR and the activities (manual bug finding and fixing) that it is designed to aid and, at least partially, replace. This will enable APR tools and techniques to be developed with a greater understanding of how they should be introduced and presented to developers.

To the best of our knowledge, we are also the only study to consider developers' instinctive, intuitive attitudes towards APR based on minimal information about how APR works. This allows us to consider developers' levels of 'dispositional trust', something that is likely to play a role in APR adoption. (We discuss the idea of 'dispositional trust' in more detail below).

III. Method

A. Introduction of APR to participants

We introduced APR to survey participants as follows: 'In recent years, there have been advances in automatic software repair techniques. Automatic software repair techniques automatically generate patches to fix bugs, often using Machine Learning or other AI techniques'. Whilst experiments tend to ask participants to respond to a specific APR tool and may provide quite a lot of information about how the tool works, we wanted to find out about developers' attitudes towards APR as a general concept. As a result, we did not want to wed ourselves to a particular APR technique paradigm. Instead, we designed the survey to introduce the notion of APR to participants as generically as possible in order to elicit developers' 'gut feelings' or 'hunches' towards APR. This corresponds to the idea of 'dispositional trust', posited by Marsh and Dibben [24] and applied, in the context of trust in automation, by Hoff and Bashir [25]. Whilst we did not explicitly ask our survey respondents about their levels of trust in APR, a lot of our survey questions probed into respondents' attitudes towards APR, where we expected trust to play a role. Considering trust in automation, Hoff and Bashir define dispositional trust as 'an individual's overall tendency to trust automation, independent of context or a specific system' and explain that 'individuals exhibit a wide variability in their tendency to trust automation' [25]. In presenting APR in generic terms, it was these kind of attitudes that we were hoping to elicit, rather than our participants' response to a specific APR technique or approach.

This is particularly important in the context of new and emerging technologies, such as APR. Krafft et al. note that 'for new technologies to be accepted, awareness of such technologies must grow and the benefits they offer must be clear, but this process can take a significant amount of time' [26]. For this process to be eased and quickened, it is important to understand developers' instinctive responses to APR, as it can help developers of APR tools consider how they present their tools to developers and increase acceptance of these tools. Given that APR is currently in an early phase of its development, with low developer awareness and low industry uptake (companies like Facebook [22] and Bloomberg [21] excepting), it is important to understand the current attitudinal baseline, levels of 'dispositional trust' [25], and what Krafft et al. term 'first impressions' [26]. Our approach also has similarities with the well-established Technology Acceptance Model (TAM), which uses constructs related to two key measures ('perceived usefulness' and 'perceived ease of use') to predict acceptance and usage of a tool [27]. Whilst we don't go so far as to suggest that attitudes can predict behaviours, like TAM we study general attitudes towards a technology with the view that such attitudes do have implications for tool adoption and usage.

The notions of 'dispositional trust' [25] or 'first impressions' [26] suggest more instinctive, intuitive and emotive responses. This is again appropriate for the study of attitudes towards an emerging technology, since much research suggests that people's attitudes towards emerging technologies are more emotional than cognitive. Loewenstein et al., for example, highlight the role of emotions in people's responses to perceived risks [28] – and emerging technologies can be included within the framework of risk. Huijts et al., in their article on public attitudes towards carbon dioxide storage techniques, suggest that emerging technologies – given their uncertainties – tend to elicit 'intuitive feelings' and activate emotions, as people struggle to weigh up risks and benefits [29]. Whilst developers do represent a more 'expert public', likely to have greater technical knowledge and understanding, Hoff and Bashir suggest that 'dispositional trust' always plays a role even when people have more knowledge (developing 'situational trust' and 'learned trust', the other two dimensions of trust identified by Marsh and Dibben) [25]. As a result, and in response to a lack of more generic human studies of APR, it was this dimension that we aimed to study.

B. Survey design

Designing the survey was a highly iterative process. We piloted the survey at various stages with three industry-based software engineers at different companies, each of whom provided detailed feedback. The pilot process was used to identify any confusing or leading questions. Participants in the pilot also helped us populate survey items where we asked survey participants to choose between multiple options (such as what made bugs particularly annoying to fix).

The final version of the survey, following this piloting process, was structured into the following sections: Part 1- Time spent bug finding and fixing; Part 2- Feelings about work in general; Part 3- Bug finding and fixing practices; Part 4- Feelings about bug finding and fixing; Part 5- Attitudes towards APR; and Part 6- Demographics.

Part 1- Time spent bug finding and fixing: This section of the survey was designed to try and capture how much time respondents spent finding and fixing bugs. We asked participants to estimate the time they had spent finding and fixing bugs the *previous day* (or their most recent day at work). One possible weakness of this question is that developers might, for example, have a particular day of the week dedicated to bug fixing. To mitigate this, we asked whether this was less than, similar to, or more than normal. We also asked participants to estimate how much time they had spent finding and fixing bugs over the previous month. All questions in this section were single-answer multiple choice.

Using specific time periods has been found in some time-use research to result in more accurate responses compared with asking about typical behaviour (for example, over 'an average week') and to also reduce cognitive loads for respondents [30] [31]. Whilst there is also some evidence in favour of asking about 'an average day' rather than 'yesterday' [30], we opted for specific recent time periods. Given the varied nature of SE activities at different times in the software development lifecycle, it might be difficult for developers to consider an 'average' period of time.

We used the phrase 'finding and fixing bugs' throughout the survey rather than 'debugging', as 'finding and fixing bugs' is a more open description of the process, whereas 'debugging' might be understood as just referring to employing a debugging tool or applying a particular debugging technique.

Part 2- Feelings about work generally: Due to the fact that we wanted to ask how participants felt about fixing and finding bugs, we also included a section that asked participants' about their feelings about work generally. This was done in order to control for the fact that people's feelings about bug finding and fixing might simply reflect their feelings towards their work more generally. Asking about feelings about work also enables us to consider whether there are any correlations between feelings about work and feelings about bug finding and fixing. In this section, respondents were asked about the extent to which their work was *challenging*, *meaningful*, *satisfying* and *frustrating*, as well as how *successful* they felt they were at work. Each question had a 5-point Likert scale response, ranging from 'never' to 'always'.

Part 3- Bug finding and fixing practices: This section of the survey combined different types of questions to try and gain understanding into developers' bug finding and fixing practices. We used open-text responses to ask developers how they found bugs, how they fixed bugs, and how they verified their fixes. We did this partly in order for developers to express their bug finding and fixing practices and strategies in their own words and partly because little research on how developers find and fix bugs exists, meaning there is little to draw from in order to establish an appropriate set of options from which developers can choose. We also included a multiple-option, multiple choice question for how respondents were alerted to the existence of a bug; a multiple choice question about whether participants mainly fixed bugs in their own code, other people's code, or both; and a grid question about how often participants fixed bugs on their own, in a pair, or in a group.

Part 4- Feelings about bug finding and fixing: In this part of the survey, we asked participants how challenging, meaningful, satisfying and frustrating they found finding and fixing bugs, as well as how successful they considered their bug fixing to be. These attitudinal descriptors were repeated from Part 2, to enable a comparison between feelings about bug finding and fixing and general feelings about work. Alongside these Likert-scale questions, we also asked a question about what made a bug specifically difficult to fix (respondents being asked to pick their top three from a series of options).

Part 5- Automatic program repair: This section introduced the notion of APR to respondents and asked a series of 5 point Likert scale questions about their general feelings about the idea of APR; their responses to a series of statements about APR; and how important they would find different aspects of a potential APR solution (for example, readability of fixes). The statements were derived from attitudinal themes uncovered in existing APR human studies. Table I shows the derivation of these statements, and the statement shorthand used in following tables in this paper.

We wanted to ask a question about what types of bugs would be most helpful for participants for an APR tool to fix. There is considerable research on fault taxonomies, classifications and categorisations within SE, but no agreed-upon model. There are also various different dimensions upon which a fault classification may be based, such as time of fault introduction (e.g. specification); effects of fault activation (e.g. data corruption); location; and type of corrective action [35]. Proposed fault classifications range from the simple (Munson and Nikora's categorisation of faults into code faults, design faults, and specification faults [36]) to far more complex. We used a semantic typology (designed for Java) from Pan et al. [37]:

- If related
- Method call related
- Sequence related
- Loop related
- Assignment related
- Switch related
- Try/catch related
- Method declaration related
- Class field related

TABLE I
APR ATTITUDINAL STATEMENTS

	Derivation	Shorthand
'Automatically generated patches would help save me time'	[18]	Time-saving
'Automatic software repair would not be able to fix complex bugs'	[32]	Not complex bugs
'I would be worried about the accuracy of automatically generated patches'	[18]	Accuracy
'I would find an automatic software repair tool useful'	[11] [33] [19] [12] [34]	Useful
'Human-written patches are more reliable than automatically generated patches'	[17]	Humans more reliable
'Automatic software repair tools might make software developers complacent'	[11]	Complacency

There were several taxonomies at this more semantic level and we chose this one for its relevance and understandability to software developers, as well as the fact that many APR approaches operate at this level. Although Pan et al. did not use their taxonomy to fix software bugs, the taxonomy has had a significant impact on APR research. Several researchers have improved and extended Pan et al.'s taxonomy and used it in APR tools [38] [39] [40]. Due to its simplicity, Pan's taxonomy has also been extended to other programming languages, e.g. JavaScript [41].

Moreover, the basis of the taxonomy is a set of constructs that might reasonably be considered the focus of developer testing for finding and fixing bugs; for example, checking boundary conditions on loops and *if* statements, range and type checks on assignments and appropriate exception handling. The taxonomy is also directed towards the object-oriented paradigm, covering method and field declaration; while we aim to be as inclusive as possible in terms of which programming language paradigm APR should be applied to, languages such as Java are attracting significant and growing industrial interest [22] [42].

Part 6- Demographics: Following advice that participants may find demographic questions off-putting if asked first [43], we included the 'about you' section of our survey at the end.

Several of our demographic questions were taken from the 2019 and 2020 Stack Overflow surveys.[1] These included the following:

- 'Including any education, how many years have you been coding?' (2020)
- 'How many years have you coded professionally (as part of your work)?' (2020)
- 'Which of the following best describes your current employment status?' (2020, with some of the options provided in the Stack Overflow survey removed, such as retired and student, as we wanted to capture responses only from developers currently working)
- 'Which of the following best describes the industry you work in? If you or your employer are involved in several industries, please choose the one most relevant to the products or services you work on' (2019)

These questions were taken from Stack Overflow to enable comparison between our own sample and theirs.

We also gathered the following demographic data: job title; size of organisation of employment; programming languages used; gender; age; and country of residence.

[1]Question guides available from https://insights.stackoverflow.com/survey

C. Validation of responses

The number of written responses we received in the open-text boxes was one indication of the validity of results (between 90.4% and 95.6%, depending on the question and generally decreasing as the survey progressed). We also checked for validity of responses by checking if, for example, there were any instances of someone ticking 'always' for all three options for the question 'how often do you find and fix bugs: alone; in a pair; in a group?'. We found two instances (0.5% of responses) where this was the case, and checked these responses thoroughly. We checked particularly for any signs that the questionnaire had been rushed through, such as repetitive answers - e.g., always ticking the same Likert scale options. We did not find any evidence of this, so did not delete these responses.

D. Sampling strategy

Sampling within SE is complex. Random samples are very difficult to achieve given a lack of census-style directories of software developers, from which a random sample could be drawn [44] [45]. Even estimates of the global population of software developers vary [46]. To establish a sample size, we used the Global Developer Population 2019 report produced by Slashdata.[2] This report estimates that there were 18.9 million developers in 2019, with 12.9 million (68.3%) of them being professional developers. The report estimates that this would have risen to 23 million by the end of 2019 (or 21.7%). We worked out the 2020 figure, at the same rate of increase, to be nearly 28 million, with 19.1 of these professional developers (based on the same breakdown of professional and hobbyist developers). Inputting this into a sample size calculator,[3] the sample required at a 95% confidence interval is 384, so we selected this as our target sample size. Wagner et al. [46] argue that a sample size close to 400 should offer 'strong generalisability', as long as checks are made for representativeness. We follow Wagner's recommendation to use large, commercial surveys to compare one's own sample. In Section IV, we discuss our survey sample demographics compared to recent Stack Overflow developer survey results. Baltes and Diehl also highlight the Stack Overflow survey as the main resource for insight into software developer demographics [45].

[2]https://www.slashdata.co/free-resources/
[3]https://www.surveysystem.com/sscalc.htm

E. Recruitment and dissemination

We used two main forms of recruitment and dissemination. Firstly, we conducted purposeful convenience sampling, targeting our partner organisations and industry contacts, as well as using relevant social media channels. We achieved 76 responses in this way. The response rate was quite slow — possibly due to Covid-related disruption to people's normal working patterns — and we realised, upon looking at the participant demographics, that our sample was skewed towards highly educated developers.

We then turned to another recruitment channel, using the online platform Prolific[4] to recruit participants. Like Amazon Mechanical Turk (AMT), Prolific participants are paid small sums for their time. However, Prolific has certain advantages over AMT. Prolific is designed specifically for academic research and allows participants to be carefully selected and filtered. Peer et al. highlight that Prolific participants are both more diverse and more honest than AMT workers [47]. Prolific is increasingly being used by researchers as a highly effective recruitment platform [47] [48], including in SE research [49].

We applied the following two filters: 'knowledge of software development techniques: debugging' and 'industry: software'. Whilst software developers work in all sorts of different industries, we felt targeting people in the software industry was the best approach to ensure that our participants were professional developers.

We found the Prolific data to be of a high standard, specifically evidenced by the fact that more Prolific participants answered the open-text questions. For example, when asked to explain their answer to 'How would you feel about using an automatic software repair tool that found and fixed bugs?' (an open-text question towards the end of the survey), 98.1% of respondents wrote an answer, compared to 71.1% of the sample we had recruited through contacts and snowballing.

F. Survey analysis

1) Statistical tests: We used two key forms of statistical test to analyse our data. Firstly, to correlate ordinal variables, the most appropriate statistical measure was Kendall's tau-b correlation coefficient. Kendall's tau-b is suitable for data that is at least ordinal. Unlike Pearson's product-moment correlation, it is non-parametric and does not assume normal distribution of data. Owing to comparing Likert-scale (ordinal) variables, our data involved many tied ranks and, as a result, it was more suitable to use Kendall's tau-b than Spearman's rank-order correlation coefficient. Another advantage of Kendall's tau is that it tests for a monotonic relationship – that is, a linear relationship, rather than assuming one.[5] We plotted jittered scatterplots to check for the existence of forms of relationship that might not be linear.

Our second key statistical test was the Chi-Square test for association, as it enables analysis of variables that are categorical (both ordinal and nominal).[6]

2) Analysis of qualitative open-text responses: For the analysis of the open-text responses, the qualitative responses were extracted and then thematically coded. The approach we took involved open coding and also negotiated agreement, as recommended by [50], using the approach detailed in [51]. Firstly, a sample was taken of the qualitative responses for each open-text question, twenty responses being chosen from each question at random. Two authors assigned codes to these responses and then discussed them, in order to negotiate agreement. This is a process of 'open coding', where codes are developed from looking at the data (an inductive approach), rather than a codebook being developed first and then applied to the data (a deductive approach). An open, inductive approach to coding was more appropriate given the exploratory nature of our research, and the fact that we were not working, for example, within an existing theoretical or conceptual framework that suggested clear categories. The codes developed through this process of negotiated agreement and open coding established a draft codebook.

We then divided the rest of the qualitative responses between the two authors to be coded independently. We applied the codes that had already been defined, but also continued an open coding approach, adding new codes as they emerged from the data. Each author flagged any new code they established for later discussion, as well as any coding they were uncertain about. The two authors then reviewed each other's codes, noting agreement, as well as coding that required discussion. Finally, we met to discuss those that required discussion, again negotiating agreement and reaching consensus on the final codes. Of the independently coded responses, there was 72.9% agreement between authors, leaving 27.1% for discussion. It should be noted that lots of instances for discussion were repeats – for example, repeat slightly differing understandings of the application of a particular code. In addition, those in need of discussion included 'partial agreement' — where, for example, there was agreement on one code but not another, or the suggestion of an additional code.

G. Replicability

We provide a replicability package at: https://github.com/winterem/APRsurvey.

IV. Our participants

A. Key participant demographics

Our participants were predominantly male (85.4%). This is less than the proportion of Stack Overflow Developer Survey professional developer respondents that identified as male in 2020 (91.7%).[7] However, Stack Overflow found that their survey was biased towards male respondents, and our own figure is likely to be nearer to the actual gender breakdown in many countries. The US Bureau of Labor Statistics, for example, estimates that women make up 20% of professional software developers. In the UK, research by Women in Tech suggests that 19% of the tech workforce are women.[8]

[4]https://www.prolific.co

[5]https://statistics.laerd.com/spss-tutorials/kendalls-tau-b-using-spss-statistics.php

[6]https://statistics.laerd.com/spss-tutorials/chi-square-test-for-association-using-spss-statistics.php

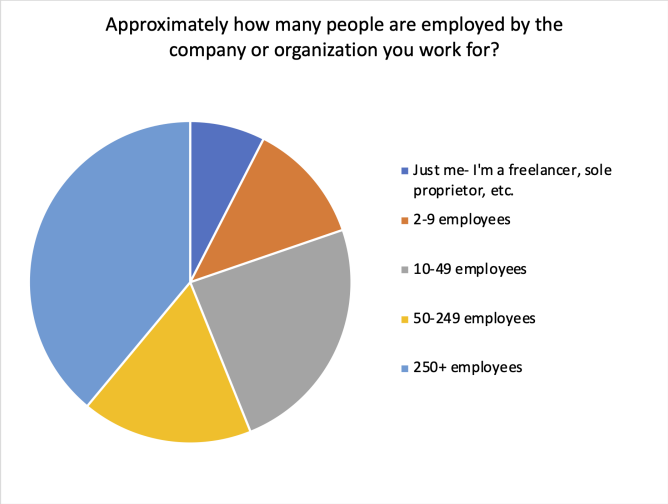
[7]All 2020 Stack Overflow statistics can be found here: https://insights.stackoverflow.com/survey/2020

[8]https://www.womenintech.co.uk/8-facts-women-tech-industry

In terms of age, just over 90% of our respondents were 44 or younger. This is very similar to the 2020 Stack Overflow respondents, of whom 90.7% of professional developers were 44 or under. The breakdown of ages is largely very similar, though we had fewer respondents in the 25-34 category (45.7% compared with 51.4%) and more respondents in the 18-24 category (25.4% compared with 17.8%). Our sample is skewed slightly younger than the Stack Overflow sample.

The three countries of residence that we had the greatest proportion of respondents from were the UK (24.6%), US (13.4%) and Portugal (11.5%). We suggest that, being based at UK institutions, potential respondents from the UK may have been more likely to recognise the names of our institutions and be inclined to fill out the survey. Like Stack Overflow, our respondents were concentrated in Europe (74.6%) and North America (20.9%), though we lack the representation from India gained by Stack Overflow.

Fig. 1. Size of company worked for



B. Education and experience

A high proportion of our sample had at least a Bachelor's degree in Computer Science or a related discipline (79.3%). This is similar to the Stack Overflow 2020 sample, with 78.1% of professional developer respondents having at least a Bachelor's degree (though the Stack Overflow study did not specify Computer Science or related, merely asking for highest formal qualification regardless of subject).

Table II shows the years spent coding professionally by our participants compared with Stack Overflow's sample.

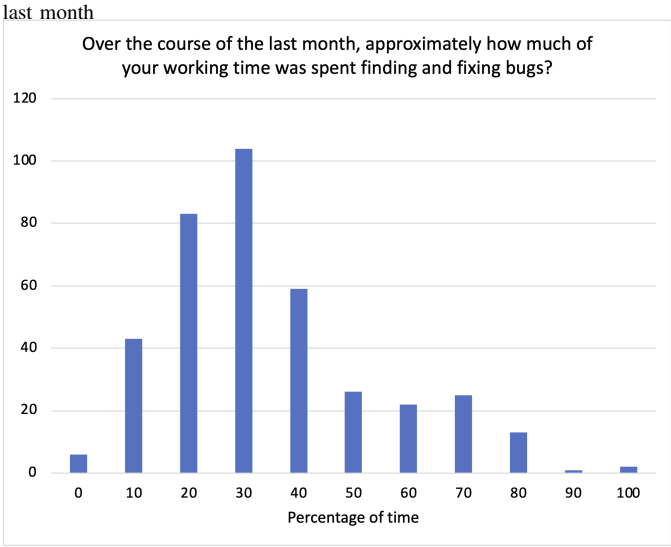
C. Employment

Just over four fifths of our participants were employed full-time, with a further 11.9% self-employed or freelance. Table 1 indicates the size of the companies participants worked for. Participants were asked to identify their sector of employment. The top three sectors were 'software development- other' (24.3%); 'information technology' (15.7%); and 'software as a service (saas) development' (11.8%). 'Software development-other' and 'information technology' were the top two sectors in the Stack Overflow 2019 survey,[9] though with lower percentages (12.0% and 10.8% respectively). Finance and banking was next, and then 'software as a service (saas) development' (7.7%).

We asked participants to rank programming languages according to which they used most at work. To calculate the most used language, we awarded three points when a language was ranked first, two points when ranked second and one point when ranked third. The language that received the highest score was Javascript, followed by Java and Python respectively, though the languages that were ranked first the most were Java, C# and Python. The 2020 Stack Overflow survey also found Javascript to be the most commonly used language among developers.

[9]The 2019 survey results can be found here: https://insights.stackoverflow.com/survey/2019

Fig. 2. Estimated percentage of time spent finding and fixing bugs over the last month



V. Findings

A. RQ1: What are software developers' current bug finding and fixing practices?

Survey respondents were asked how long they had spent finding and fixing bugs the previous day (or their most recent day at work). The results are shown in Figure 3, showing that over half of participants spent less than one hour finding and fixing bugs, with a further quarter spending between one and two hours. Table 2 shows the time over the last month that participants estimated they had spent finding and fixing bugs, the modal category being 30%, a significant proportion of developer time, though less than has often been suggested (see [52]).

We also asked developers whether their time spent bug finding and fixing the previous day was less than normal, the same, or more than normal. Table III shows the results. For participants who spent less than 30 minutes fixing bugs, the

TABLE II
YEARS SPENT CODING PROFESSIONALLY

Years	Percentage (and cumulative %) for our sample	Percentage (and cumulative %) for 2020 Stack Overflow sample
4 or less	51.9%	36.6%
5 to 9	19.2% (71.1%)	26.8% (63.4%)
10 to 14	12.7% (83.8%)	14.7% (78.1%)
15 to 19	6.5% (90.3%)	7.6% (85.7%)
20 to 24	4.2% (94.5%)	6.0% (91.7%)
25 to 29	2.3% (96.8%)	2.4% (94.1%)
30 to 34	1.8% (98.8%)	1.6% (95.7%)
35 to 39	0.5%	0.8%
40 to 44	0.5%	0.4%
45 to 49	0.0%	0.1%
50 or over	0.3%	0.1%

Fig. 3. Time survey respondents spent finding and fixing bugs the previous day

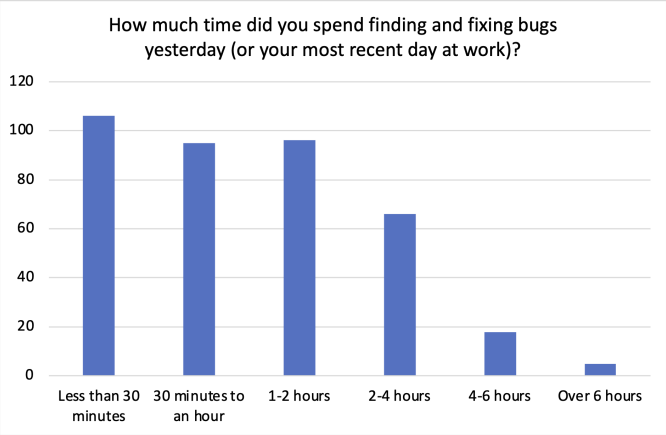
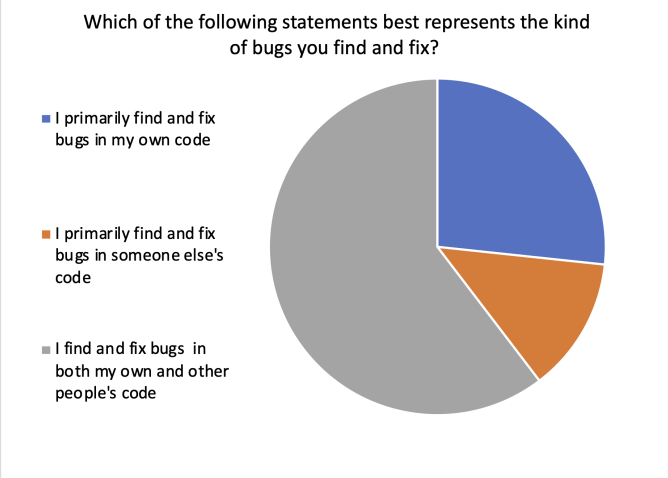


Fig. 4. Do developers mainly fix bugs in their own code or other people's?



modal category was 'less than normal'. For the rest of the time options, the modal category was 'the same', until the highest option for time spent (over 6 hours), for which the modal category was 'more than normal'. Other than at the extremes, the developers' previous working days seem to have been mostly typical in terms of time spent finding and fixing bugs.

Figure 4 shows in whose code developers are fixing bugs, indicating that only a small proportion of developers primarily fix bugs in other people's code. Respondents were also asked the extent to which they found and fixed bugs alone, or with others. Respondents fixed bugs alone 'always' in 22.5% of cases, and 67.8% fixed bugs alone 'most of the time'. Bug finding and fixing seems to be predominantly a solitary activity.

Over three quarters of our participants (77.6%) stated that test cases for the code they worked on were designed manually, rather than automatically. This demonstrates that automated testing is not being used extensively in industry. While we do not know why this is, it may indicate a reluctance to use automated tools in industry or that the tools do not work according to developers' needs.

The responses to the open-text question 'how do you fix bugs?' indicated that the majority of participants do *not* use specific tools (e.g. debuggers) to approach bug fixing. We received 355 written responses to this question; of these, 286 (80.6%) indicated no use of any specific tool, compared with 69 responses that mentioned use of a specific tool to aid fixing bugs. This confirms other research findings [1] [5] that the use of debugging tools is not widespread among professional developers, indicating that the take-up of tools is slow and uneven in industry, as well as research on barriers to adoption of static analysis tools [53]. The qualitative responses also indicate a fairly manual and haphazard approach to bug fixing, for example, *'mostly trial and error'*, *'thinking about how things work in certain scenarios and trying [a] few modifications on the code until everything works fine'*, and *'keep changing the code until it works usually'*.

In answer to RQ1 (*What are software developers' current bug finding and fixing practices?*), we find that bug finding and fixing is a predominantly solitary task for developers, with most developers spending up to 2 hours fixing bugs in a 'normal day'. For most developers, this task also seems to be mainly manual, with test cases designed manually in the majority of cases and very few developers using debugging

TABLE III
HOW NORMAL WAS THE PREVIOUS DAY FOR TIME SPENT FINDING AND FIXING BUGS?

	Less than normal	The same	More than normal	Total
Less than 30 minutes	53 (53.5%)	43 (43.4%)	3 (3.0%)	99
30 minutes to 1 hour	32 (34%)	49 (52.1%)	13 (13.8%)	94
1-2 hours	13 (13.8%)	51 (54.3%)	30 (31.9%)	94
2-4 hours	3 (4.5%)	32 (48.5%)	31 (47.0%)	66
4-6 hours	2 (11.1%)	6 (33.3%)	10 (55.6%)	18
Over 6 hours	0	2 (40.0%)	3 (60.0%)	5

Fig. 5. Satisfaction levels- work and finding and fixing bugs

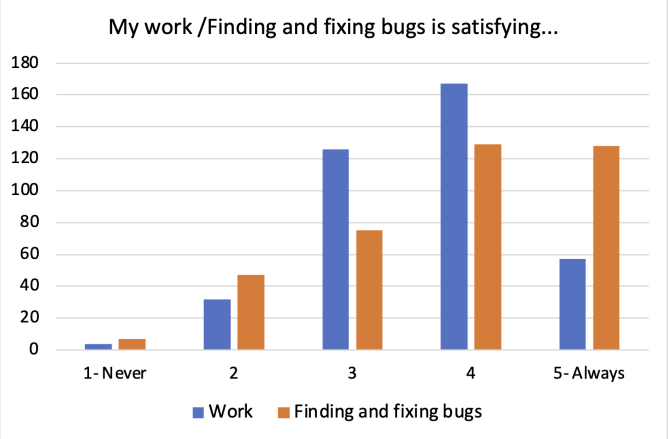
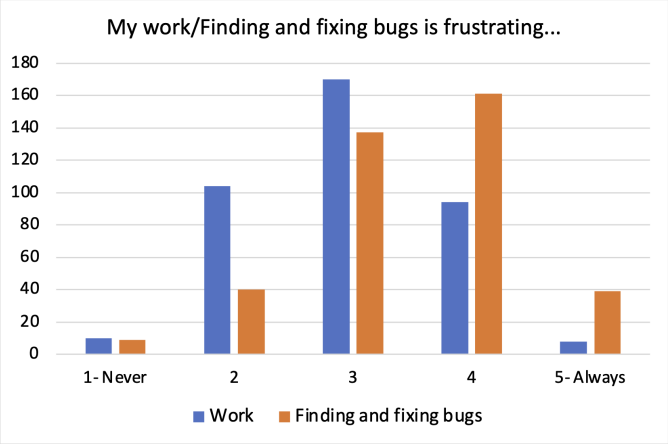


Fig. 6. Frustration levels- work and finding and fixing bugs



tools. We therefore find empirical confirmation for what is often claimed in the APR literature - that bug fixing is time-consuming and often manual.

B. RQ2: How do software developers feel about bug finding and fixing?

Table IV shows the results for how developers feel about bug finding and fixing. This demonstrates that more developers consider finding and fixing bugs 'always satisfying' (33.2% of respondents) than 'always frustrating (10.1%).

The variables 'satisfying' and 'frustrating' were particularly interesting regarding the relationship between work generally and finding and fixing bugs. This is demonstrated in Figures 5 and 6. Bug finding and fixing emerge as both more satisfying *and* more frustrating than work generally. It seems that bug fixing elicits more extreme reactions (*both* positive *and* negative) than work generally.

We performed Chi-Square tests of association to see if there was an association between feelings about finding and fixing bugs and the following variables: whose code the bugs were in; the degree to which respondents found and fixed bugs alone; and participants' highest qualification in Computer Science. The Chi-square test was appropriate for these variables as they are nominal. We found no evidence of any statistically significant association, meaning that feelings about bug finding and fixing do not seem to be influenced by fixing bugs alone, CS qualifications, or whose code the bugs were in.

We also computed Kendall's tau-b coefficients to check for coefficients between feelings about finding and fixing bugs and age, years coding professionally, and years coding including education, to see whether experience had any impact. We only found four Kendall's tau-b coefficients that were greater than 0.1 and statistically significant; these are shown in Table V. The coefficients are not high — none of them are greater than 0.2 — so they are indicative only of a weak relationship. The main association shown in Table V is that older developers, and developers with more years coding experience, are *slightly* more likely to see themselves as successful at bug finding and fixing.

We then considered the relationship between feelings about work and feelings about bug finding and fixing, in order to see whether feelings about bug fixing are correlated with general work feelings. We again used Kendall's tau-b as both these variables are ordinal. Here we found evidence of slightly stronger relationships. There were four relationships with a coefficient greater than 0.3. These were: work-meaningful and finding and fixing bugs-meaningful; work-satisfying and finding and fixing bugs-meaningful; work-frustrating and finding and fixing bugs-frustrating; and work-successful and finding and fixing bugs-successful. This suggests a relationship between feelings about work and feelings about finding and fixing bugs, implying that to some moderate degree developers are likely to have similar feelings about both work and finding and fixing bugs. This implies that it is not as simple as developers having certain feelings about finding and fixing bugs, but that these feelings may be influenced by broader work environment factors — attitudes towards bug finding and

TABLE IV
HOW DEVELOPERS FEEL ABOUT FINDING AND FIXING BUGS

	1- Never	2	3	4	5 -Always
Finding and fixing bugs is challenging	0.5%	5.7%	34.7%	41.2%	17.9%
Finding and fixing bugs is meaningful	0.8%	10.1%	25.1%	36.3%	27.7%
Finding and fixing bugs is satisfying	1.8%	12.2%	19.4%	33.4%	33.2%
Finding and fixing bugs is frustrating	2.3%	10.4%	35.5%	41.7%	10.1%
My bug finding and fixing is successful	0.0%	1.8%	24.7%	56.9%	16.6%

TABLE V
KENDALL'S TAU-B COEFFICIENTS GREATER THAN 0.1. NB: * = SIGNIFICANT AT THE 0.05 LEVEL; ** = SIGNIFICANT AT THE 0.01 LEVEL

	Finding and fixing bugs is frustrating	My bug finding and fixing is successful
Years coding professionally	N/A	0.164**
Years coding including education	N/A	0.174**
Age	-0.109*	0.101*

fixing do not exist in isolation.

We asked developers, using open-text response questions, what they liked *most* and *least* about finding and fixing bugs. We coded 355 responses about what participants liked *most* about finding and fixing bugs, having removed from the data set all blanks and any responses that were too vague or unclear for us to confidently assign a theme to them. Only 9 respondents (0.01%) said that they liked **nothing** about finding and fixing bugs, though a further 25 responses (7.0%) were coded as **'finishing it'**, meaning that the thing these respondents liked most about finding and fixing bugs was the completion of the task, suggesting little inherent value in the task itself.

152 responses (42.8%) were thematically coded as **'satisfaction'**, indicating that these participants found some element of finding and fixing bugs satisfying. The sense of **challenge** was highlighted in 53 responses (14.9%), for example, *'I like the challenge of discovering and fixing bugs'* and *'I like the constant challenge that it provides'*. In addition, 45 responses (12.7%) were coded as **'learning'**, meaning that participants felt that they improved their knowledge, understanding or skills through finding and fixing bugs. Examples include: *'I can learn from my own mistakes and gain valuable experience'*; *'learning why something doesn't work the way it is expected to'*; *'it improves my skills as a coder'*; and *'digging through the code and becoming super familiar with how it works so that it's less of a black box'*.

Several responses were related to the impact of finding and fixing bugs. 34 responses (9.6%) were coded as **'having impact'** – this refers to finding and fixing bugs having a beneficial impact on, for example, clients or customers. 104 responses (29.3%) were coded as **'contributing to a working or improved codebase/system/product'**. For the theme 'having impact', the following are indicative quotations: *'I make a better product that more users will enjoy'*; *'providing value to end users'*; *'making customers happy'*; and *'solving an issue that bothered stakeholders'*. For the theme 'contributing to a working or improved codebase/system/product', examples include:

- 'Knowing that the product is verifiably better now than it was before'
- 'It improves the code quality and it is extremely satisfying to fix bugs and know that the code is better than it was previously'
- 'I like the feeling of improving our product and making it more robust'
- 'Satisfaction of fixing products back to the desired state'

For what respondents liked *least* about finding and fixing bugs, we thematically coded 357 responses. The most frequently occurring theme was **'time'**, 113 responses (31.7%) being assigned this theme. 'Time' refers to bug finding and fixing taking a lot of time, for example *'it often takes a lot of time'* and *'the time involved'*. There were several other thematic codes related to time, such as **'takes time away from other activities'** (14 responses; 3.9%) and **'time pressures'** (12 responses; 3.4%).

The next most frequently occurring theme was **'finding'**, with 110 responses (30.8%) being assigned this theme. This demonstrates that many developers have a particular problem with finding where a bug is in the code and/or what is causing the bug. As one participant expressed it, *'the process of finding; fixing is alright'*. Other responses included:

- 'The finding process itself is kind of boring'
- 'I don't like the frustration of identifying bugs that are hard to find'
- 'The long search for bugs, especially when the edge case is extremely obscure and only happens on specific hardware/software configurations on the client'

50 distinct responses (14.0%) were coded with themes related to the **nature of the code base**, including **other people's code** (30 responses; 8.4%), **poorly written code** (16 responses; 4.5%) and **poorly documented code** (11 responses; 3.1%). Examples of this category include:

- 'Working on archaic code that I have little knowledge of that has been written according to old practices'
- 'Some people's code is sloppy and difficult to parse'
- 'If the code is badly organised or badly documented, it gets hard and frustrating to find a solution'

TABLE VI
KENDALL'S TAU-B COEFFICIENTS GREATER THAN 0.1. NB: * = SIGNIFICANT AT THE 0.05 LEVEL; ** = SIGNIFICANT AT THE 0.01 LEVEL

	Bugs- challenging	**Bugs- meaningful**	**Bugs- satisfying**	**Bugs- frustrating**	**Bugs- successful**
Work- challenging	0.284**	0.138**	N/A	N/A	N/A
Work- meaningful	N/A	0.357**	0.126**	-0.140**	0.195**
Work- satisfying	N/A	0.305**	0.206**	N/A	0.236**
Work- frustrating	0.165**	-0.126**	N/A	0.313**	-0.151**
Work - successful	N/A	0.130**	N/A	-0.132**	0.365**

TABLE VII
MOST FREQUENTLY OCCURRING THEMATIC CODES FOR THE SURVEY QUESTION 'WHAT DO YOU LIKE MOST ABOUT FINDING AND FIXING BUGS?' NB: RESPONSES COULD BE TAGGED WITH MULTIPLE THEMATIC CODES

Thematic code	**Percentage of responses**	**Indicative quotation**
'Satisfaction'	42.8%	'It is extremely satisfying to fix bugs'
'Contributing to a working or improved codebase/system/product'	29.3%	'Knowing that the product is verifiably better now than it was before'
'Challenge'	14.9%	'I like the challenge of discovering and fixing bugs'
'Learning'	12.7%	'It improves my skills as a coder'
'Having impact'	9.6%	'Providing value to end users'

TABLE VIII
MOST FREQUENTLY OCCURRING THEMATIC CODES FOR THE SURVEY QUESTION 'WHAT DO YOU LIKE LEAST ABOUT FINDING AND FIXING BUGS?' NB: RESPONSES COULD BE TAGGED WITH MULTIPLE THEMATIC CODES

Thematic code	**Percentage of responses**	**Indicative quotation**
'Time'	31.7%	'It often takes a lot of time'
'Finding'	30.8%	'The process of finding; fixing is alright'
'Frustration'	22.7%	'It can be quite frustrating to debug'
'Difficulty'	18.5%	'It can be frustrating and difficult to debug'
'Boring, tedious or unrewarding work'	14.0%	'They [the bugs] are usually so small that it's not even rewarding to fix them'

There was also a cluster of thematic codes connected to respondents' experiences of, and feelings about, bug finding and fixing. 81 responses (22.7%) were coded as **'frustration'** – respondents found some element of bug finding and fixing frustrating. 66 responses (18.5%) were coded as **'difficulty'**, meaning that participants found at least some part of bug finding and fixing hard. Difficulty was related most often to finding bugs (38 responses). Finally, 50 responses (14.0%) stated that the thing that they liked least about finding and fixing bugs was that it was **boring, tedious or unrewarding work**, for example *'looking at lines of code can get very boring'*; *'they [the bugs] are usually so small that it's not even rewarding to fix them*; *'find and fix very irrelevant bugs that don't impact the code/project'*; and *'it's not delivering direct value and rarely something to be super proud of'*.

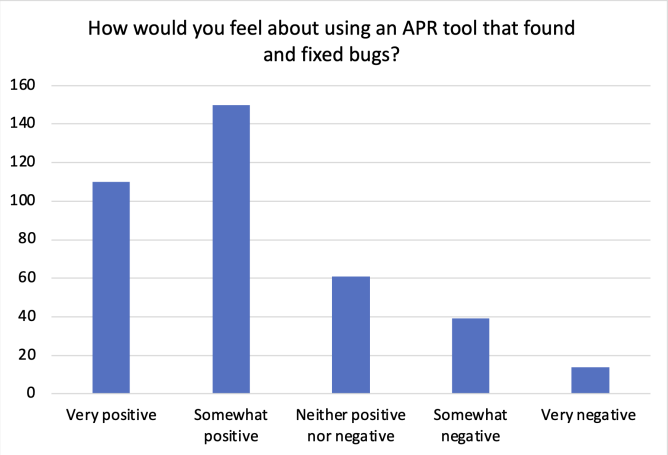
For the question about what makes a bug particularly annoying to fix, participants were asked to rank a series of options in first, second and third place. Most often ranked first (91 participants) was 'when it's in very complex code', followed by 'when it's in poorly documented code' (76 participants) and 'when it's in very old code' (71 participants). If all three rankings are taken into account and scored (so being ranked in first place gains 3 points, etc.), 'when it's in very complex code' remains the number one factor, followed by 'when it's in very old code' and 'when it's in poorly documented code'.

To summarise, in answer to RQ2 (*How do software developers feel about bug finding and fixing?*), we find that developers' attitudes towards finding and fixing bugs are complex. Developers find bug-finding and -fixing *both* more satisfying *and* more frustrating than they do their work generally. We found very few statistically significant relationships with demographic variables, showing that experience has very little impact on how developers experience bug fixing. However, we do find some evidence for a connection between feelings about finding and fixing bugs and feelings about work generally, suggesting that developers' feelings about bug fixing do not exist in isolation from general work conditions. The qualitative data demonstrates that some key sources of satisfaction found in bug fixing are a sense of improving the codebase, system or product being developed, a sense of challenge, and and gaining understanding, skills and knowledge.

C. RQ3: What are software developers' instinctive feelings towards APR?

Figure 7 shows respondents' general feelings towards APR, while Table IX shows the preferences of developers for different kinds of APR options. Figure 7 demonstrates that the modal category of participants considered themselves 'somewhat positive' about using an APR tool. Table IX indicates a strong preference among developers to remain part of the process, with 88.0% favouring an APR tool that provides developers with different fixes to choose from over a tool

Fig. 7. General attitudes towards APR



that provides developers with fixes to approve and a tool that automatically applies fixes.

We also asked developers to rank when an APR tool would be most useful to them, shown in Table X. Unsurprisingly, developers have a strong preference for tools finding bugs as soon as possible (i.e. during development). We asked a similar question about the usefulness of APR at different points in the software development process, shown in Table XI. Testing was seen as the most useful point for APR to be used, followed by implementation. This is perhaps surprising, as we might expect APR to be considered more useful to developers in implementation rather than testing, as it could enable developers to take advantage of APR earlier in the process.

We also asked how far respondents agreed with a series of statements about APR, the results being shown in Table XIV. This table shows that the most agreed with statement was Accuracy- 'I would be worried about the accuracy of automatically generated patches', with 51.9% of respondents strongly agreeing with this statement. The least agreement was with the Complacency statement- 'Automatic software repair tools might make software developers complacent', with only 12.9% of respondents strongly agreeing with this statement. Table XV shows how important respondents rated a series of possible features for an APR tool. Correctness of patches was seen as the most important feature of an APR tool (66.1% of respondents considered this 'extremely' important), followed by understandability/readability of patches (54.2%) and human verification of patches (51.6%). Participants considered speed of patch generation the least important (11.6% of respondents considered this 'extremely' important), suggesting that developers may see a trade-off between speed and accuracy. This is in contrast to Noller et al.'s survey [23], in which developers stated that they want an APR tool to generate patches within 30-60 minutes.

We also asked participants how useful they would find an APR tool to fix bugs in different types of coding structure. The results are shown in XVI, and indicate that developers would find most use in an APR tool that fixed bugs in loop-related coding structures and in if-related coding structures. Both of these types of coding structure are fault-prone and complex. This demonstrates that developers would find fixes to complex bugs useful (as they are likely the most time-consuming for developers to fix manually), but, as seen above, developers are also sceptical as to APR's ability to provide fixes for more complex bugs. It is also worth noting that we had a high non-response rate (18%) for this question. This may reflect non-response from software developers that do not use Java or related languages, but it may also indicate that developers found it difficult to contemplate APR in less abstract and more specific terms due to a lack of understanding of APR's capabilities.

Again, we carried out statistical tests, testing for an association between highest computer science qualification and all of the attitudinal variables shown in Tables I, IX, and XV using the Chi-Square test (as appropriate for nominal variables). We did this in order to consider whether there was any relationship between CS qualifications and attitudes towards APR. We found no evidence of any statistically significant relationships, except for an association between qualifications and the statement 'automatic software repair might make developers complacent'. This was significant where p is less than 0.05, allowing us to reject the null hypothesis of no association between these variables. The cross tabulation, however, demonstrates no straightforward association between highest Computer Science (CS) qualification and agreement with the Complacency statement. The greatest difference between expected and observed values is found for respondents with no CS qualification and respondents with a Bachelor's degree. Respondents with no CS qualification at degree level were *less* likely to agree with the statement 'automatic software repair might make developers complacent', while respondents with a BSc (or equivalent) in CS were more likely to agree. So, those with no CS qualification are *more* likely to think APR will cause developer complacency.

We also used Kendall's tau to test whether there was any relationship between respondents' age and experience levels (both years coding professionally and years coding including education) and their general attitude towards APR; their response to a series of statements about APR; and what respondents consider important in an APR tool. For general attitude towards APR, we found no evidence of any relationship with age and experience levels. The Kendall's tau-b for each of these pairs of variables was less than 0.1, with 0 equalling no relationship. For responses to attitudinal statements about APR, we found only two instances of Kendall's tau-b coefficients that were greater than 0.1 and indicative of a very weak relationship. These were both for the statement 'human-written patches are more reliable than automatically generated patches'. For years coding professionally and age, the Kendall's tau-b coefficients were -0.110 and -0.147 respectively, indicating a weak negative relationship between agreeing with this statement and years of experience coding and age. This implies that older and more experienced developers are *slightly* less likely to see automatically generated patches as more reliable than human-written ones. Both of these coefficients are statistically significant at the 0.01 level.

TABLE IX
PERCENTAGE OF RESPONDENTS WHO RANKED DIFFERENT APR OPTIONS AS THEIR FIRST, SECOND AND THIRD CHOICES

	Ranked first	Ranked second	Ranked third
'An APR tool that automatically applies fixes'	4.7%	32.1%	63.2%
'An APR tool that provides developers with fixes to approve'	7.3%	62.4%	30.3%
'An APR tool that provides developers with different fixes to choose from'	88.0%	5.5%	6.5%

TABLE X
WHEN WOULD AN APR TOOL BE USEFUL?

	Ranked first	Ranked second	Ranked third
'Bugs found during development'	162	67	53
'Bugs found during testing'	58	186	39
'Bugs found post-release'	2	30	191

TABLE XI
HOW USEFUL WOULD YOU FIND AN APR TOOL DURING THE FOLLOWING PARTS OF THE SOFTWARE DEVELOPMENT PROCESS?

	Extremely useful	Very useful	Moderately useful	Slightly useful	Not at all useful	Don't know
Specification	4.5%	7.9%	12.6%	18.2%	50.0%	6.8%
Requirements	4.7%	9.2%	14.2%	19.0%	46.2%	6.6%
Design	6.1%	12.7%	17.5%	24.1%	34.7%	5.0%
Implementation	25.5%	35.5%	24.7%	10.5%	2.4%	1.3%
Testing	38.9%	36.3%	15.3%	6.1%	1.6%	1.6%

TABLE XII
MOST FREQUENTLY OCCURRING THEMATIC CODES FOR RESPONDENTS WHO WERE 'VERY POSITIVE' OR 'SOMEWHAT POSITIVE' ABOUT APR. NB: THEMATIC CODES ARE ORDERED ACCORDING TO 'VERY POSITIVE' RESPONDENTS, BUT NOTE THE DIFFERENCES FOR THOSE THAT WERE 'SOMEWHAT POSITIVE'. RESPONSES COULD BE TAGGED WITH MULTIPLE THEMATIC CODES

Thematic code	Percentage of responses	Indicative quotation
General positivity	Very positive – 73.1%; Somewhat positive – 25.9%	'Heaven sent'
APR would make job easier/reduce workload	Very positive – 49.5%; Somewhat positive – 22.3%	'This would make my life a lot easier'
APR would free up time for other activities	Very positive – 22.9%; Somewhat positive – 11.5%	'More time for development of new things'
Concern, uncertainty, conditionality	Very positive – 26.9%; Somewhat positive – 74.1%	'I would be sceptical of its efficacy until proven'

Exploring the relationship between age and experience and what developers saw as important in an APR tool, we found seven Kendall's tau-b coefficients that were greater than 0.1 and indicative of a very weak relationship. Table XVII highlights these seven values. Years experience and age were slightly correlated with seeing correctness of patches as more important and speed of patch generation as less important.

In summary, we found very weak relationships between the demographic variables relating to age and experience and attitudes towards APR.

The open-text responses for attitudes towards APR were particularly illuminating. Whilst almost 70% of respondents were 'very positive' or 'somewhat positive' about using an APR tool, the open-text responses (participants having been asked to explain their answer) demonstrate a high degree of concern and scepticism. Of a total of 260 respondents that said they were 'very positive' or 'somewhat positive' about APR, 238 provided a response in the open-text box.

We coded 232 written responses from participants who had stated that they were either 'very positive' (93 responses) or 'somewhat positive' (139 responses) about APR. There was a considerable difference in the qualitative response between those who stated they were 'very positive' and those who were 'somewhat positive'. Of those who said they were 'very positive' about APR, 46 responses (49.5%) felt that **APR would make their job easier or reduce their workload**, while 21 (22.9%) said **APR would free up their time for other activities**. For those who were 'somewhat positive' about APR, 31 responses (22.3%) felt that APR would make their job easier or reduce their workload and 16 responses (11.5%) that APR would free up time for other activities. Examples of 'making job easier/reduced workload' include *'I think it would make bug fixing faster and more relaxed'*; *'this would make my life a lot easier'*; and *'it would save me a lot of time and effort'*. Examples of 'more time for other activities' are *'more time for development of new things'*; *'programmers can spend more time on the design of software to create more robust software'*; and *'anything which promotes*

TABLE XIII
MOST FREQUENTLY OCCURRING THEMATIC CODES FOR RESPONDENTS WHO PICKED 'AN APR TOOL THAT PROVIDES DEVELOPERS WITH DIFFERENT FIXES TO CHOOSE FROM' AS THEIR TOP CHOICE. NB: RESPONSES COULD BE TAGGED WITH MULTIPLE THEMATIC CODES

Thematic code	Percentage of responses	Indicative quotation
Need for human judgement/review	47.0%	'Letting the programmer decide is always the best option'
Importance of having choice	24.9%	'Can imagine many situations where a bug may have many different solutions depending on the desired behaviours. The developer needs to choose an appropriate fix, and if the tool only presents one, it limits its usefulness'
Distrust	23.0%	'I don't trust an AI to choose the right fix'
Control	18.0%	'I would like to retain full control of my code'

TABLE XIV
LEVELS OF AGREEMENT WITH DIFFERENT STATEMENTS ABOUT APR. NB: PLEASE SEE TABLE I FOR FULL STATEMENTS AND THEIR DERIVATION.

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
Time-saving	37.0%	39.9%	11.7%	8.8%	2.7%
Not complex bugs	38.9%	31.5%	17.3%	9.6%	2.7%
Accuracy	51.9%	36.9%	6.4%	4.2%	0.6%
Useful	35.5%	42.8%	14.6%	6.0%	1.1%
Humans more reliable	23.4%	24.2%	40.0%	11.3%	1.1%
Complacency	12.9%	37.3%	23.2%	16.5%	10.1%

TABLE XV
THE IMPORTANCE OF DIFFERENT FEATURES IN AN APR TOOL

	Extremely	Very	Moderately	Slightly	Not at all
Understandability/readability of patches	54.2%	31.2%	7.9%	5.6%	1.1%
Human verification of patches	51.6%	32.3%	11.6%	4.0%	0.5%
Full automation (i.e. humans out of the loop)	3.5%	4.6%	15.7%	28.5%	47.7%
Fit of tool within current workflow	25.9%	42.6%	19.2%	10.8%	1.5%
Correctness of patches	66.1%	22.7%	7.6%	2.3%	1.3%
Speed of patch generation	11.6%	19.5%	41.7%	20.8%	6.3%
Test results for the patches	34.5%	41.1%	17.6%	5.0%	1.8%
Similarity of generated patches to human written patches	18.2%	32.4%	28.4%	12.1%	8.8%

TABLE XVI
HOW USEFUL WOULD APR BE FOR FIXING BUGS IN DIFFERENT TYPES OF CODING STRUCTURE

	Extremely	Very	Moderately	Slightly	Not at all
If related	32.4%	29.8%	20.5%	14.1%	3.1%
Method-call related	28.9%	30.2%	24.7%	12.0%	4.2%
Loop related	38.1%	28.8%	22.8%	8.1%	2.2%
Assignment related	23.4%	27.6%	24.0%	19.8%	5.2%
Switch related	19.5%	28.7%	27.4%	19.5%	4.9%
Try-catch related	27.2%	29.1%	23.8%	15.0%	5.0%
Method declaration related	20.7%	20.7%	23.4%	25.1%	10.0%
Sequence related	22.0%	21.6%	28.2%	22.3%	5.8%
Class field related	16.0%	24.6%	25.3%	26.3%	7.8%

TABLE XVII
KENDALL'S TAU-B COEFFICIENTS GREATER THAN 0.1. NB: FOR THESE VARIABLES, EXTREMELY IMPORTANT WAS SCORED AS 1 AND NOT AT ALL IMPORTANT AS 5. * = SIGNIFICANT AT THE 0.05 LEVEL; ** = SIGNIFICANT AT THE 0.01 LEVEL

	Understandability/readability of patches	Correctness of patches	Speed of patch generation
Years coding professionally	N/A	-0.110**	0.136**
Years coding including education	N/A	-0.120**	0.123**
Age	-0.101*	-0.150**	0.119**

efficiency, reliability and quality is great. If it allows humans to focus on creativity and user experience so much the better'.

We coded as **'general positivity'** all responses that were completely positive: this was 73.1% of 'very positive' responses and 25.9% of 'somewhat positive' responses. The rest of the responses (26.9% for 'very positive' and 74.1% for 'somewhat positive') had a degree of concern, uncertainty, or conditionality. For those responses that were 'somewhat positive', 51 (36.7%) were thematically coded as **'uncertainty'**, 17 (12.2%) as **'distrust'**, 29 (20.9%) as **'conditionality'** and 61 responses (43.9%) specified some form of concern. The most common concern was **feasibility** – 37 responses (26.6%). Examples of this include *'I would be sceptical of its efficacy until proven'*; *'I don't think that we are at the state where AI is smart enough to find and fix bugs in complex system'*; *'I would be happy if it worked, but I doubt it'*; and *'I can't understand how the software will be able to fix the error without knowing the final goal of the code'*. The next most common concern was **'unintended consequences'** (16 responses; 11.5%). Examples are *'I've seen the experiments that were done with auto generated pull requests on git hub and the results looked really exciting, but I remember it misunderstanding the code occasionally and reintroducing old bugs back into the codebases'*; and *'I'm not sure how successful it would be and [it] could introduce other bugs'*. These qualitative responses indicate a more complex picture than the statistics taken alone, suggesting that, although our respondents stated that they were positive about APR, their attitude can be summarised more, as one participant expressed it, as 'cautiously optimistic'.

The open-text responses about their APR preference (automatic application of fixes, verification of fix, or choice between fixes) are also helpful for understanding what is at stake for developers with the potential introduction of APR. We coded 217 responses that had listed their number one preference as choosing between fixes. The most prevalent theme was **'need for human judgement/review'**, which applied to 102 responses (47.0%). Examples of this include:

- 'Letting the programmer decide is always the best option'
- 'I still think human intervention would be vital to ensure nothing is missed or done incorrectly'
- 'I prefer to have the final say as the developer'
- 'Manual approval of fixes is a MUST because I wouldn't want to risk any of the code getting messed up'

54 responses (24.9%) mentioned the **importance and desirability of having choice**. Answers coded as this include:

- 'If an automatic tool is only capable of providing one fix, it is useless. If an automatic tool can provide multiple fixes to choose from, it at least allows engineers to debate the merits and drawbacks of each approach before committing to something'
- 'I can imagine many stations where a bug may have many different solutions depending on the desired behaviours. The developer needs to choose an appropriate fix, and if the tool only presents one, it limits its usefulness'
- 'People may get lazy and just rubber stamp the fixes so it would be good to provide a variety of fixes so someone can think about which is best and avoid the tendency to rubber stamp things'

The need for **control** was tagged for 39 responses (18.0%), Examples include *'I would like to retain full control of my code'*; *'choosing from various fixes helps me to maintain the control of the code'*; *'developers should always have control'*; and *'I prefer having control over the code that I'm expected to be accountable for in terms of quality'*. 20 responses (9.2%) were thematically coded as **'need to be informed'**, such as *'developers should always know when code is being modified'*.

Distrust was a theme for 50 responses (23.0%), such as *'I don't trust an AI to choose the right fix'*; *'I don't believe an automatic system could be trusted without oversight'*; and *'I've seen enough automated tools to know they're not trustable'*. There were also several concerns, the most common being **'tool lacks contextual understanding'** (29 responses; 13.4%) and **'unintended consequences'** (20 responses; 9.2%). Examples of responses coded as 'tool lacks contextual understanding' include *'the tool cannot fully understand the business logic or why certain things have been coded in a specific format'*; *'I don't think the automated tool could understand the goals priorities and context to fix complex bugs without developer input'*; and *'the tool does not know what the intended functionality of the program is'*. Examples of responses coded as 'unintended consequences' include:*'it's important to be able to review bug fixes in order to predict any other effects they [the fixes] may have on the wider codebase'* and *'if [the fix was] complex, I wouldn't be confident that the change doesn't have unintended side effects'*.

In answer to RQ3 (*What are software developers' instinctive feelings towards APR?*), we find that developers are mostly positive about APR, but have a strong preference for a tool that allows them to remain 'in the loop'. Accuracy of automatically generated patches is a key concern for developers, and they are much less interested in how quickly patches are generated. Again, we find evidence of very few statistically significant relationships between attitudes towards APR and key demographic variables. The qualitative data demonstrates that, even amongst developers that are positive about APR, there are many concerns, and that having control is a significant factor that informs developers' preference for a tool that enables them to stay in the loop.

VI. Discussion and recommendations

A. There is a clear need for APR

Our survey confirms other studies in demonstrating that developers spend a significant amount of time finding and fixing bugs. We also find that bug finding and fixing is mainly a manual activity, with few developers utilising tooling to assist them. Though we present several caveats in this paper regarding how APR should be presented to developers, the time spent fixing bugs and the lack of tooling currently being used confirm that there is an important space for APR to fill in terms of removing workload and providing tooling to developers. We also find a good degree of positivity and interest from developers, though there are several concerns that need to be addressed.

B. Bug fixing is not all bad

We find no evidence of an overwhelmingly negative attitude to bug fixing among developers. More developers in fact found bug finding and fixing 'always satisfying' than 'always frustrating'. Far from the 'universal bad' of bug fixing presented in the APR literature (as something boring, tedious and annoying), we in fact find that developers find significant satisfaction in finding and fixing bugs, as well as bug fixing being a source of learning and development.

This has important implications for the APR research community. How, for example, might developers who experience finding and fixing bugs as satisfying and rewarding be persuaded to adopt APR tools? What tasks might we emphasise as APR freeing up developer time to work on instead? Automatically generated fixes could also be a source of knowledge for software developers, but this depends upon what additional information is presented about a bug and its automatically generated fix.

C. Developers are cautiously optimistic about APR

Whilst 69.5% of developers were 'very positive' or 'somewhat positive' about 'using an APR tool that found and fixed bugs', the qualitative responses indicate that it is not necessarily as simple as this. Instead the qualitative responses demonstrate that, despite a generally positive outlook towards APR, developers still had many concerns and caveats. Given the uncertainty of developers' 'dispositional trust', much work will be needed to develop 'situational' and 'learned' trust. However, we do not find opposition towards APR amongst developers, suggesting there is clear enthusiasm among the developer community for the introduction of APR tools and techniques.

D. What impacts upon developer attitudes?

Our survey finds very scant evidence of a relationship between attitudes towards APR and key demographic variables, such as age, education and experience. This raises significant questions regarding what factors may in fact impact upon developer attitudes towards APR. We suggest that these factors may be things that are difficult to measure, such as organisational or team culture. However, future research is needed to explore what factors may play a role.

The qualitative responses also suggest that developer values play a significant role in attitudes towards APR, with 'control' (linked to human agency and self-direction) a frequently occurring reason for why developers preferred an APR tool that allowed them to check or choose from fixes over an APR tool that applied fixes automatically.

E. Recommendations for APR tool design

APR design should continue to emphasise developer-in-the-loop systems: Our results demonstrate that developers are currently reluctant to embrace a fully automated APR system, instead preferring a system where they would either be offered a fix to check and apply, or a series of fix options to choose between. This is in line with much APR research that assumes that developers will be involved in reviewing and approving patches (for example, [54]). However, some recent research, for example on automating correctness assessment for patches generated by program repair tools [55] demonstrates innovation that could contribute to fully automated APR systems. Our survey results suggests that — at present at least — such systems are likely to face considerable barriers to adoption by professional developers.

The emphasis that developers in our survey placed upon having control also means that careful consideration needs to be given to how we design APR user experiences so that they feel fully part of the process.

Patches should be readable and/or accompanied by information to aid understanding: Our results show that understandability/readability of fixes is a key concern for developers and something that should be taken into consideration in APR research. Whilst readability of fixes has been stressed in some APR research (for example, research that generates fixes learnt from human-written fixes [17]), other APR research has stressed the importance of 'alien' fixes that may be very different to human-written ones and less understandable to developers (for example, [56]). In the case of such 'alien' fixes, our research suggests that thought needs to be given to what kind of information accompanies automatically generated fixes so that developers can gain understanding of the fix. This should also help with some of the issues related to trust highlighted above.

APR research should consider the benefits of providing multiple fixes to a single bug: Our survey respondents had a strong preference for choosing between multiple fixes. There is some APR research that takes this approach (for example, [57], [58], [59]) – our results suggest that this may be a promising future direction, although more work is needed to identify what the ideal number of fixes for developers to choose from is.

VII. Threats to validity

A. External validity

Our sample size of 386 is large enough to be considered broadly representative of the software developer population. We have also compared the demographics of our sample with Stack Overflow's survey demographics, demonstrating reasonable similarity. There were, however, some differences between our sample and Stack Overflow's, particularly the more European geographic basis of our sample and the lack of input from developers in India. We do not consider it likely that geographic location has a significant impact on attitudes towards bug fixing and APR. However, it may be that cultural workplace factors do play some role, so future research would be welcome to explore whether the findings from our predominantly European and North American sample hold up in other cultural settings. We hope our replication package will enable this survey to be repeated in diverse contexts. Like all research samples that rely upon participant consent, ours is a volunteer sample. Volunteer samples raise questions as to the potential characteristics of people who are likely to volunteer to participate in a research study. However, little research has explored the potential impact of volunteer samples on the validity of results.

One other threat pertaining to generalisability is the fact that the most straightforward way to recruit practising software developers from Prolific was to specify 'software industry' as place of work. This may have excluded from our participation pool software developers working in industries whose main focus is something different, and may explain, for example, the low amount of participants in our survey from 'finance and banking' compared to Stack Overflow's sample. Whilst our sample is large, it may be that it doesn't generalise to specific domains in which software developers work. One possible direction for future work is to consider specific domains, especially those that are safety- or security-critical.

Survey research, like any method, has inherent limitations. Whilst our survey was predominantly attitudinal, we did ask some questions more related to behaviours, such as time spent finding and fixing bugs. It is likely that these self-reported measures may be different to the actual time that could, for example, be identified through observations or screen-recording techniques. To avoid too much bias from self-reporting, we used the proxies of education and years spent coding professionally to stand in for experience, rather than using self-reported experience levels. It is also important to note that developers' stated attitudes do not necessarily map onto behaviours, as there is a complex relationship between values and actions.

B. Internal validity

Whilst we find evidence in our results of some (weak) associations between variables, we do not test for relationships of causation, as this is an exploratory study of an under-studied area.

In terms of the analysis of results, we chose appropriate statistical tests for our data, based on the nature of our data (unlikely to be normally distributed, and involving many tied ranks due to Likert-scale variables). Statistical tests were performed using dedicated statistics software (SPSS), to limit the possibility of error. There are some possible threats to statistical validity, such as the difficulty of establishing non-linear relationships. Though we plotted jittered scatterplots to check for this, the amount of tied ranks made this challenging. Using 7-point, rather than 5-point, Likert scales might have allowed more granularity of results, but could also have been more challenging for participants to answer.

The qualitative analysis was conducted by two authors in order to mitigate the effects of interpretive subjectivity. We also used a combination of independent coding and discussion to reach negotiated agreement. The independent coding of each qualitative response was reviewed by the other author, meaning that each response was dual-coded.

C. Construct validity

We piloted our survey with software developers before disseminating it more broadly to check that our questions could be understood by developers and used vocabulary that was clear and relatable. Despite this piloting process, we found some evidence that certain questions were perhaps not clearly understood by developers, demonstrated by blank responses. Specifically, 8.5% and 8.0% of respondents respectively did not state their agreement with the statements 'human-written patches are more reliable than automatically generated patches' and 'automatic software repair tools might make software developers complacent'. We had a particularly high non-response rate for the question about different kinds of bug (e.g. if-related). Approximately 18% of respondents did not respond to this question, probably because the constructs were Java-biased and not so understandable for developers coding in other languages.

One threat to construct validity is the combination of bug finding and fixing in some of our questions, such as the Likert scale attitudinal questions. Whilst this was not raised as an issue in our pilot study with developers, the qualitative responses to 'what do you like most/least about finding and fixing bugs?' demonstrate that at least some developers draw a distinction between the two activities. This may mean that it was challenging for some developers to answer the Likert scale questions, if their feelings towards bug finding and bug fixing are different. However, there were high response rates for the Likert scale items, which we might not have expected if these questions had been too confusing for developers to answer.

VIII. Conclusions and further work

To our knowledge, this is the first survey that asks developers about their feelings about both finding and fixing bugs *and* automatic program repair, considering attitudes to both APR and the activity (manual bug finding and fixing) APR is designed to minimise or replace. We find that bug finding and fixing is *both* a satisfying *and* frustrating activity for developers, rather than something solely frustrating. This is significant because it means that APR desires to replace an activity (manual bug finding and fixing) that developers in fact do derive some value from, including skill and knowledge development. Understanding the human (developer) aspects of coding practices and using that understanding to inform practice has often been overlooked in the past; the study presented provides insights into how developers perceive the bug fixing and repair process.

When it comes to developers' attitudes towards APR, our findings present both challenges and opportunities for the further development of APR tools and techniques. Developers are largely positive about the idea of APR, demonstrating an enthusiasm and appetite for APR tools, but also 'cautiously optimistic', with the existence of several concerns and caveats. Our data suggests that attention should be given to developing APR user interfaces in which developers remain in the loop and in which presented fixes are understandable to developers. Developers have often criticised the lack of (and quality of) tools that they use in their day to day coding work. In fact, providing usable tools for developers is a problem that both industry and academia (and collaboration efforts) have struggled to tackle in the past [60]. Results from our survey support the call for proper developer involvement and support, especially in an APR context.

Our data also demonstrates very few statistically significant relationships between developer attitudes towards APR

and key demographic variables, such as age, experience and education. This raises significant questions as to what factors may in fact influence attitudes towards APR, and future work exploring the potential impact of, for example, workplace culture would be valuable. The socio-technical aspects of coding and development in general has been a popular research topic since the 1970s. Yet, a body of knowledge about developer demographics and the impact they can have *in the field* does not seem to have materialised. Perhaps empirical studies of developers have too often focused on students or small numbers of industry participants and key factors such as developer experience have been largely overlooked in those studies.

Given that APR is an emergent and quickly growing field, the results from our survey are important in informing the future of APR tool design and development that meets developer needs. Our survey findings suggest that APR tools that keep the developer in the loop and prioritise understandability of patches are likely to have stronger industry uptake. The advent of technologies such as APR has made addressing the issue of what developers *really* feel and want more important, not less.

Acknowledgment

This work is funded by the Engineering and Physical Sciences Research Council, UK (grant number EP/S005749/2). We are very grateful to our anonymous survey participants for taking part in this research.

References

[1] M. Beller, N. Spruit, D. Spinellis, and A. Zaidman, "On the dichotomy of debugging behavior among programmers," in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 572–583. [Online]. Available: https://doi.org/10.1145/3180155.3180175

[2] M. Böhme, E. O. Soremekun, S. Chattopadhyay, E. Ugherughe, and A. Zeller, "Where is the bug and how is it fixed? an experiment with practitioners," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 117–128. [Online]. Available: https://doi.org/10.1145/3106237.3106255

[3] W. Weimer, "Program repair, patch quality, and human factors," May 2021, keynote at 2nd International Workshop on Automated Program Repair (APR 2021).

[4] E. R. Winter, V. Nowack, D. Bowes, S. Counsell, T. Hall, S. O. Haraldsson, and J. Woodward, "Let's talk with developers, not about developers: A review of automatic program repair research," *IEEE Transactions on Software Engineering*, pp. 1–1, 2022.

[5] M. T. e. a. M. Perscheid, B. Siegmund, "Studying the advancement in debugging practice of professional software developers," *Software Quality Journal*, vol. 25, pp. 83–110, 2017.

[6] X. Kong, L. Zhang, W. E. Wong, and B. Li, "The impacts of techniques, programs and tests on automated program repair: An empirical study," *Journal of Systems and Software*, vol. 137, pp. 480–496, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121217301279

[7] Y. Lou, A. Ghanbari, X. Li, L. Zhang, H. Zhang, D. Hao, and L. Zhang, "Can automated program repair refine fault localization? a unified debugging approach," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 75–87. [Online]. Available: https://doi.org/10.1145/3395363.3397351

[8] W. Zou, D. Lo, Z. Chen, X. Xia, Y. Feng, and B. Xu, "How practitioners perceive automated bug report management techniques," *IEEE Transactions on Software Engineering*, 2018.

[9] Z. Wan, X. Xia, A. E. Hassan, D. Lo, J. Yin, and X. Yang, "Perceptions, expectations, and challenges in defect prediction," *IEEE Transactions on Software Engineering*, pp. 1–26, 2018.

[10] P. S. Kochhar, X. Xia, D. Lo, and S. Li, "Practitioners' expectations on automated fault localization," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ser. ISSTA 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 165–176. [Online]. Available: https://doi.org/10.1145/2931037.2931051

[11] B. Daniel, V. Jagannath, D. Dig, and D. Marinov, "ReAssert: Suggesting Repairs for Broken Unit Tests," in *Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering*, 2009, pp. 433–444.

[12] S. Kaleeswaran, V. Tulsian, A. Kanade, and A. Orso, "Minthint: Automated Synthesis of Repair Hints," in *Proceedings of the International Conference on Software Engineering*, 2014, pp. 266–276.

[13] Y. Tao, J. Kim, S. Kim, and C. Xu, "Automatically Generated Patches As Debugging Aids: a Human Study," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 64–74.

[14] A. Kalyanpur, B. Parsia, E. Sirin, and B. Cuenca-Grau, "Repairing Unsatisfiable Concepts in OWL Ontologies," in *The Semantic Web: Research and Applications*, 2006, vol. 4011, pp. 170–184.

[15] J. Yi, U. Ahmed, A. Karkare, S. Tan, and A. Roychoudhury, "A feasibility study of using automated program repair for introductory programming assignments," in *Proceedings of ESEC/FSE*, 2017.

[16] J. P. Cambronero, J. Shen, J. Cito, E. Glassman, and M. Rinard, "Characterizing developer use of automatically generated patches," in *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2019, pp. 181–185.

[17] D. Kim, J. Nam, J. Song, and S. Kim, "Automatic Patch Generation Learned From Human-Written Patches," in *Proceedings of ICSE*, 2013.

[18] C. Liu, J. Yang, L. Tan, and M. Hafiz, "R2Fix: Automatically Generating Bug Fixes From Bug Reports," in *Proceedings of the International Conference on Software Testing, Verification and Validation (ICST)*, 2013, pp. 282–291.

[19] H. Hata, E. Shihab, and G. Neubig, "Learning to generate corrective patches using neural machine translation," *arXiv preprint 1812.07170*, 2018.

[20] C. Parnin and A. Orso, "Are automated debugging techniques actually helping programmers?" in *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, ser. ISSTA '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 199–209. [Online]. Available: https://doi.org/10.1145/2001420.2001445

[21] S. Kirbas, E. Windels, O. McBello, K. Kells, M. Pagano, R. Szalanski, V. Nowack, E. R. Winter, S. Counsell, D. Bowes, T. Hall, S. Haraldsson, and J. Woodward, "On the introduction of automatic program repair in bloomberg," *IEEE Software*, vol. 38, no. 4, pp. 43–51, 2021.

[22] J. Bader, A. Scott, M. Pradel, and S. Chandra, "Getafix: Learning to fix bugs automatically," *Proc. ACM Program. Lang.*, vol. 3, no. OOPSLA, Oct. 2019. [Online]. Available: https://doi.org/10.1145/3360585

[23] Y. Noller, R. Shariffdeen, X. Gao, and A. Roychoudhury, "How to trust auto-generated code patches? A developer survey and empirical assessment of existing program repair tools," *CoRR*, vol. abs/2108.13064, 2021. [Online]. Available: https://arxiv.org/abs/2108.13064

[24] S. Marsh and M. R. Dibben, "The role of trust in information science and technology," *Annual Review of Information Science and Technology*, vol. 37, no. 1, pp. 465–498, 2005. [Online]. Available: https://doi.org/10.1002/aris.1440370111

[25] K. A. Hoff and M. Bashir, "Trust in automation: Integrating empirical evidence on factors that influence trust," *Human Factors*, vol. 57, no. 3, pp. 407–434, 2015, pMID: 25875432. [Online]. Available: https://doi.org/10.1177/0018720814547570

[26] M. F. Krafft, K.-J. Stol, and B. Fitzgerald, "How do free/open source developers pick their tools? a delphi study of the debian project," in *Proceedings of the 38th International Conference on Software Engineering Companion*, ser. ICSE '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 232–241. [Online]. Available: https://doi.org/10.1145/2889160.2889248

[27] F. D. Davis, "Perceived usefulness, perceived ease of use, and user acceptance of information technology," *MIS Quarterly*, vol. 13, no. 3, pp. 319–340, 1989. [Online]. Available: http://www.jstor.org/stable/249008

[28] C. K. H. G. F. Loewenstein, E. U. Weber and N. Welch, "Risk as feelings," *Psychological Bulletin*, vol. 127, no. 2, pp. 267–286, 2001. [Online]. Available: https://doi.org/10.1037/0033-2909.127.2.267

[29] C. J. H. M. Nicole M. A. Huijts and A. L. Meijnders, "Social acceptance of carbon dioxide storage," *Energy policy*, vol. 35, no. 5, pp. 2780–2789, 2007.

[30] T. Araujo, A. Wonneberger, P. Neijens, and C. de Vreese, "How much time do you spend online? understanding and improving the accuracy of self-reported measures of internet use," *Communication Methods and Measures*, vol. 11, no. 3, pp. 173–190, 2017.

[31] L. Chang and J. A. Krosnick, "Measuring the frequency of regular behaviors: Comparing the "typical week" to the "past week"," *Sociological Methodology*, vol. 33, no. 1, pp. 55–80, 2003.

[32] V. Balachandran, "Fix-it: An extensible code auto-fix component in review bot," in *IEEE 13th International Working Conference on Source Code Analysis and Manipulation, SCAM 2013*, 2013, pp. 167–172.

[33] S. Gulwani, I. Radiček, and F. Zuleger, "Automated clustering and program repair for introductory programming assignments," in *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2018, pp. 465–480.

[34] J. Lee, D. Song, S. So, and H. Oh, "Automatic diagnosis and correction of logical errors for functional programming assignments," *Proceedings of OOPSLA*, 2018.

[35] J. Ploski, M. Rohr, P. Schwenkenberg, and W. Hasselbring, "Research issues in software fault categorization," *SIGSOFT Softw. Eng. Notes*, vol. 32, no. 6, 2007. [Online]. Available: https://doi.org/10.1145/1317471.1317478

[36] C. Masuck, "Categorizing faults in the software build cycle decreases the total number of faults," *J. Comput. Sci. Coll.*, vol. 21, no. 2, p. 19–26, Dec. 2005.

[37] K. Pan, S. Kim, and E. J. Whitehead, "Toward an understanding of bug fix patterns," *Empirical Softw. Engg.*, vol. 14, no. 3, p. 286–315, Jun. 2009. [Online]. Available: https://doi.org/10.1007/s10664-008-9077-5

[38] K. Liu, A. Koyuncu, D. Kim, and T. F. Bissyandé, "Tbar: Revisiting template-based automated program repair," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 31–42. [Online]. Available: https://doi.org/10.1145/3293882.3330577

[39] M. Martinez, L. Duchien, and M. Monperrus, "Accurate Extraction of Bug Fix Pattern Occurrences using Abstract Syntax Tree Analysis," Inria, Technical Report hal-01075938, 2014. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01075938

[40] S. K. Nath, R. Merkel, and M. F. Lau, "On the improvement of a fault classification scheme with implications for white-box testing," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, ser. SAC '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 1123–1130. [Online]. Available: https://doi.org/10.1145/2245276.2231953

[41] P. Gyimesi, B. Vancsics, A. Stocco, D. Mazinanian, Á. Beszédes, R. Ferenc, and A. Mesbah, "Bugsjs: a benchmark and taxonomy of javascript bugs," *Software Testing, Verification and Reliability*, vol. n/a, no. n/a, p. e1751, 2020, e1751 stvr.1751. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/stvr.1751

[42] A. Marginean, J. Bader, S. Chandra, M. Harman, Y. Jia, K. Mao, A. Mols, and A. Scott, "Sapfix: automated end-to-end repair at scale," in *International Conference on Software Engineering, Montreal, Canada, 2019*. IEEE / ACM, 2019, pp. 269–278.

[43] L. B. Bourque, *How to conduct self-administered and mail surveys*, ser. The survey kit ; 3. Thousand Oaks: Sage, 1995.

[44] R. M. de Mello and G. H. Travassos, "Surveys in software engineering: Identifying representative samples," in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2961111.2962632

[45] S. Baltes and S. Diehl, "Worse than spam: Issues in sampling software developers," in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2961111.2962628

[46] S. Wagner, D. Méndez Fernández, M. Felderer, D. Graziotin, and M. Kalinowski, "Challenges in survey research," 2019.

[47] E. Peer, L. Brandimarte, S. Samat, and A. Acquisti, "Beyond the turk: Alternative platforms for crowdsourcing behavioral research," *Journal of Experimental Social Psychology*, vol. 70, pp. 153 – 163, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022103116303201

[48] S. Palan and C. Schitter, "Prolific.ac—a subject pool for online experiments," *Journal of Behavioral and Experimental Finance*, vol. 17, pp. 22 – 27, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2214635017300989

[49] D. Russo and K. Stol, "Gender differences in personality traits of software engineers," *IEEE Transactions on Software Engineering*, pp. 1–1, 2020.

[50] M. Hilton, N. Nelson, T. Tunnell, D. Marinov, and D. Dig, "Trade-offs in continuous integration: Assurance, security, and flexibility," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 197–207. [Online]. Available: https://doi.org/10.1145/3106237.3106270

[51] J. L. Campbell, C. Quincy, J. Osserman, and O. K. Pedersen, "Coding in-depth semistructured interviews: Problems of unitization and intercoder reliability and agreement," *Sociological Methods & Research*, vol. 42, no. 3, pp. 294–320, 2013. [Online]. Available: https://doi.org/10.1177/0049124113500475

[52] S. Akbarinasaji, B. Caglayan, and A. Bener, "Predicting bug-fixing time: A replication study using an open source software project," *Journal of Systems and Software*, vol. 136, pp. 173–186, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121217300365

[53] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?" in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 672–681.

[54] C. Le Goues, M. Dewey-Vogt, S. Forrest, and W. Weimer, "A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each," in *2012 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 3–13.

[55] H. Ye, M. Martinez, and M. Monperrus, "Automated patch assessment for program repair at scale," *Empirial Software Engineering*, vol. 26, no. 20, pp. 1–38, 2021. [Online]. Available: https://doi.org/10.1007/s10664-020-09920-w

[56] M. Monperrus, "A critical review of "automatic patch generation learned from human-written patches": Essay on the problem statement and the evaluation of automatic software repair," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 234–242. [Online]. Available: https://doi.org/10.1145/2568225.2568324

[57] A. Ghanbari and L. Zhang, "Prapr: Practical program repair via bytecode mutation," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019, pp. 1118–1121.

[58] C.-P. Wong, P. Santiesteban, C. Kästner, and C. Le Goues, *VarFix: Balancing Edit Expressiveness and Search Effectiveness in Automated Program Repair*. New York, NY, USA: Association for Computing Machinery, 2021, p. 354–366. [Online]. Available: https://doi.org/10.1145/3468264.3468600

[59] L. Chen, Y. Pei, and C. A. Furia, "Contract-based program repair without the contracts: An extended study," *IEEE Transactions on Software Engineering*, vol. 47, no. 12, pp. 2841–2857, 2021.

[60] D. Spinellis, "Version control systems," *IEEE Softw.*, vol. 22, no. 5, pp. 108–109, 2005. [Online]. Available: https://doi.org/10.1109/MS.2005.140

Emily Winter is a Lecturer in the School of Computing and Communications at Lancaster University, specialising in the socio-technical aspects of Software Engineering. A sociologist by background (PhD, Lancaster University, 2017), her interests are centered around the perceptions and attitudes of software developers about the technologies that they build and the tools that they use.

David Bowes is a Senior Lecturer in Computer Science at the Lancaster University. David has developed significant expertise in analysing defects in software over a period of over ten years and published widely in the area of defect prediction. He is an expert in software development and brings a focus on the production of successful tools. He has previously developed tools to collect data, analyse defective code, and assess the performance of defect prediction models. David has a deep knowledge of analysis methods, having built many defect prediction models.

Steve Counsell is a Professor of Software Engineering in the Department of Computer Science at Brunel and Head of the Brunel Software Engineering Laboratory (BSEL). His PhD is from the University of London (2002) and he has published over 190 research papers on topics including data mining, software refactoring, software evolution and defect analysis. He is a Fellow of the British Computer Society and was a software developer in industry prior to academia. Steve has worked extensively on large research projects with industry in the past.

Tracy Hall is a professor with Lancaster University. Her research interests include software engineering, code analysis and defect prediction. Contact her at tracy.hall@lancaster.ac.uk; https://www.lancaster.ac.uk/scc/about-us/people/tracy-hall

Saemundur O. Haraldsson is a Lecturer at the University of Stirling. He has co-organised every tutorial on Genetic Improvement at GECCO, PPSN, and CEC. He has co-authored multiple publications on the subject, including two that have received best paper awards the first comprehensive survey on GI which was published in 2017. He has been invited to give talks on the subject in multiple venues for academical, industrial, and general public audiences worldwide. His PhD thesis (submitted in May 2017) details his work on the world's first live GI integration in an industrial application.

Vesna Nowack received the PhD degree in Computer Architecture from Universitat Politècnica de Catalunya, Spain, in 2016. She became a teaching assistant at Technische Universität Dresden, Germany in 2017. Since June 2019, she has been a postdoctoral researcher at Queen Mary University of London, UK. Her current research focuses on automated program repair, in particular genetic improvement, generation of fix patterns and application of repair tools in industry.

John R. Woodward received the B.Sc. degree in theoretical physics, M.Sc. degree in cognitive science, and Ph.D. degree in computer science, all from the University of Birmingham, U.K. He is currently with School of Electronic Engineering and Computer Science at Queen Mary University of London U.K, where he is the head of the Operational Research Group. Previously he was with the European Organization for Nuclear Research (CERN), Switzerland, where he conducted research into particle physics, the Royal Air Force as an Environmental Noise Scientist, and Electronic Data Systems as a Systems Engineer.