# Let's Talk *With* Developers, Not *About* Developers: A Review of Automatic Program Repair Research

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

**Abstract**—Automatic program repair (APR) offers significant potential for automating some coding tasks. Using APR could reduce the high costs historically associated with fixing code faults and deliver significant benefits to software engineering. Adopting APR could also have profound implications for software developers' daily activities, transforming their work practices. To realise the benefits of APR it is vital that we consider how developers feel about APR and the impact APR may have on developers' work. Developing APR tools without consideration of the developer is likely to undermine the success of APR deployment. In this paper, we critically review how developers are considered in APR research by analysing how human factors are treated in 260 studies from Monperrus's Living Review of APR. Over half of the 260 studies in our review were motivated by a problem faced by developers (e.g., the difficulty associated with fixing faults). Despite these human-oriented motivations, fewer than 7% of the 260 studies included a human study. We looked in detail at these human studies and found their quality mixed (for example, one human study was based on input from only one developer). Our results suggest that software developers are often talked *about* in APR studies, but are rarely talked *with*. A more comprehensive and reliable understanding of developer human factors in relation to APR is needed. Without this understanding, it will be difficult to develop APR tools and techniques which integrate effectively into developers' workflows. We recommend a future research agenda to advance the study of human factors in APR.

**Index Terms**—Human factors, software development, automatic program repair

## 1 Introduction

We investigate how the human factors associated with developers are considered in automatic program repair (APR) research, and, where human studies have been conducted, we evaluate their quality. APR is an increasingly important area of software engineering research, being the first relatively mature application of automatic code generation. Automatically fixing code faults promises many potential benefits, including improving the quality of software [1], reducing the development time spent on fault fixing [2] and lowering the overall costs of software development [3]. These benefits are very attractive to software companies with several (e.g., Facebook) already trying out APR techniques in their development pipeline [4].

APR is distinct from many of the other tools used by developers. Whilst other tools offer developers assistance in their work, APR has the potential to remove the developer entirely from the process of bug fixing. Realising the benefits of APR techniques requires a significant transformation of software developers' working practices. For successful APR exploitation, developers need to accept the automation of their previously manual fixing tasks and embrace and use new APR tools and techniques. Developers will need to change some of their day-to-day tasks. Some tasks may be removed (for example, reducing manual fault-finding and fault-fixing) and replaced with others (for example, providing APR tools with specifications or verifying automatically generated patches). Fully automated APR tools may lead to developer workloads being restructured in unknown and possibly unpopular ways, given the freeing up of time previously taken up with manual code repair. Overall, APR tools and techniques are likely to disrupt the workflow and working practices of software developers, and may also impact developer job satisfaction, motivation and retention. Developing APR tools and techniques that are acceptable to developers is critical to successfully capitalising on the benefits that APR promises. Given the potential of APR to disrupt software developers' workflows it is essential that the significant socio-technical implications of APR are

- *Emily Winter, Vesna Nowack, David Bowes, and Tracy Hall are with the School of Computing and Communications, Lancaster University, LA1 4YW Bailrigg, Lancaster, U.K. E-mail: {e.winter, d.h.bowes, tracy.hall}@lancaster.ac.uk, v.nowack@qmul.ac.uk.*
- *Steve Counsell is with the Department of Computer Science, Brunel University of London UB8 3PH Uxbridge, London, U.K. E-mail: steve.counsell@brunel.ac.uk.*
- *Sæmundur Haraldsson is with the Department of Computing Science and Mathematics, University of Stirling, FK9 4LA Stirling, U.K. E-mail: saemundur.haraldsson@stir.ac.uk.*
- *John Woodward is with the School of Electronic Engineering and Computer Science, Queen Mary University of London, E1 4NS London, U.K. E-mail: j.woodward@qmul.ac.uk.*

*Manuscript received 14 October 2021; revised 1 February 2022; accepted 2 February 2022. Date of publication 16 February 2022; date of current version 9 January 2023.*
*This work was supported by the Engineering and Physical Sciences Research Council, U.K. under Grant EP/S005749/2.*
*(Corresponding author: Emily Winter.)*
*Recommended for acceptance by P. Runeson.*
*Digital Object Identifier no. 10.1109/TSE.2022.3152089*

considered in APR research. For example, there is a risk that developers feel threatened by APR, resisting the technology's adoption. Resistance is common when new technologies replace humans in the work place (e.g., [5], [6]) and requires great effort to overcome. The interplay between APR and developers must be carefully understood and APR technology offered to developers appropriately. To avoid APR being rejected by developers, research effort must be targeted at understanding the human factors related to APR tools.

In his 2020 keynote [7], Westley Weimer (one of the originators of APR) highlighted the urgent need to better understand the human factors associated with developer use of APR. We respond directly to Weimer's call for greater understanding by providing a comprehensive analysis of the current state-of-the-art in APR human factors research. We hope our analysis will equip future researchers with a helpful base on which to motivate and design human studies in APR. Our results should enable the acceleration of APR human factors research and help generate a clearer understanding of how APR tools should be designed for positive interaction with human developers.

Our aim is to investigate the extent to which human factors are taken into account by existing APR research. We critically review APR studies reported in the literature. Our review is similar to a systematic literature review [8] except that we use an existing corpus of APR studies rather than identifying those studies ourselves. We analyse 260 APR studies curated in Monperrus's Living Review on Automatic Program Repair [9] in terms of their consideration of developer human factors. We identify whether APR studies are motivated by problems faced by software developers and whether studies claim that the APR tool or technique being proposed is helpful to software developers. We looked in detail at any research that included a human study, evaluating the quality of these human studies to get a better understanding of the maturity and reliability of existing knowledge related to human factors in APR.

The results of our review of APR research suggest that there is considerable work still to be done in understanding APR in relation to developers. Of 186 Living Review papers that introduced a new tool or technique, 65% were motivated by a problem that software developers face (for example, the difficulty of fixing faults) and 34% claimed that their tool or technique would be helpful to software developers (for example, by saving developer time), despite presenting no human study to demonstrate this. Less than 7% of studies in the Living Review (17 of 260) include some form of human study. Our quality assessment finds that the quality of these 17 studies varied, with few justifying key study design decisions, such as why particular sampling strategies or data collection methods were used. Overall, our research suggests that automatic program repair studies frequently talk *about* developers, but very rarely talk *with* developers.

Successfully transferring APR into industrial practice depends on APR techniques complementing, rather than disrupting, developers' working practices and workflows. Our findings suggest that significantly more APR research needs to examine the developer human factors, and the quality of human studies must improve if APR technical progress is to be effectively exploited by industry. Despite Software Engineering being recognised as both a social and technical activity [10], [11], technical innovation continues to dominate Software Engineering research [12]. Neglecting the social and human dimension of technical innovation is not unique to APR and underpins the lack of practitioner uptake that has plagued much Software Engineering research. For example, most debugging tools developed by software engineering researchers have not been adopted widely by professional software developers [13]. APR is a relatively new area of Software Engineering. As such we now have a chance to address the socio-technical balance in future research. We advocate embedding consideration of developers alongside technical innovation to enable the development of tools and techniques likely to transfer into industrial practice. On the basis of our review, we develop and present a future research agenda to advance the study of human factors in APR.

Our research aim to advance the study of human factors in APR has certain caveats. It is not our recommendation that *all* studies in APR include a user study. There are many cases where this would not be appropriate and could in fact slow down the pace of innovation in APR. It is important that innovative approaches be published, though they may not be at a stage where a user study would be appropriate. Instead, we simply encourage *more* APR researchers to carry out thorough and well-motivated user studies, both 'scoping studies' that explore developer needs and user evaluations. Perhaps counter-intuitively, we caution against user studies being seen as prerequisites for publishing research results. Research carried out by Buse *et al.* found that 'highly selective conferences tend to publish a larger proportion of papers containing user evaluations' [14]. Our quality assessment of the human studies in APR suggests that this may result in poorly-designed user studies tacked onto the end of technical papers as a way of fulfilling the expectations of peer review. We would encourage a shift within the SE community, away from *expecting* a user study as a form of validation — which may allow for the publication of poor user studies — to instead having higher standards for user studies when they are presented.

This paper is structured as follows: Section 2 outlines related work; Section 3 highlights our aims, research questions and contribution; and Section 4 describes our methodology. Sections 5 and 6 provide findings and discussion. In Section 7 we make recommendations for future developer-centred research in APR. Before concluding (Section 9), we identify some threats to validity and action taken to mitigate them (Section 8).

## 2 Related Work

Reviews of secondary literature – including systematic literature reviews, annotated bibliographies and various kinds of literature synthesis – are common in software engineering. There have been two surveys specifically of the APR literature. Gazzola, Micucci and Mariani [2] surveyed 108 papers, which they categorised into different types of APR, such as the different kinds of algorithms deployed. Monperrus's bibliography [15] similarly divides work on APR into its different technical approaches or types. Neither of these reviews identifies papers that include a human study, although in their section on 'empirical evidence', Gazzola *et al.* mention one study with developers. Within APR, there have also been more specific reviews, such as Liu *et al.*'s critical review on how APR systems are evaluated [16]. In addition, there have been several systematic literature reviews of different human factors in software engineering, such as motivation [17] and personality [18], as well as a more overarching systematic literature review of 'behavioural software engineering' [19].

Our work can be broadly positioned alongside such studies. However, systematic literature reviews and synthesis studies tend to cover more established areas of research. For example, research synthesis aims to compare and contrast evidence from multiple sources in order 'to build knowledge and reach conclusions about the empirical support for a phenomenon' [20]. By contrast, human factors are understudied in APR and not yet an established field of research, making many synthesis approaches unsuitable. Similarly to Liu *et al.*'s recent study that provided a critical review on the evaluation of APR systems [16], we take a critical and evaluative approach to assess a less developed research space. Our goal is to understand the current state of research in human factors in APR and how such research can be advanced. Liu *et al.*'s study considered the metrics used in the evaluation of APR systems (for example, number of correct patches generated, time needed to generate patches), and does not examine the role of human studies in APR evaluation. However, our research supports their finding that it is difficult to compare APR tools and techniques due to widely varying methods of evaluation.

Our work also has similarities in approach to Storey *et al.*'s [12] review of human aspects in the publications that appeared at ICSE and in the *Empirical Software Engineering* journal during a one year snapshot (2017). Storey *et al.* focus on *who* benefits from a research study, *what* the main type of research contribution is and *how* the research was carried out. We similarly consider whether the APR papers we review introduce a new tool/technique and, in our quality assessment of the existing human studies, identify the research methods used. However, we also provide an in-depth quality assessment of existing human studies within APR, in order to understand the current state of the field and identify potential research gaps. This enriches the focus of Storey *et al.* of 'how much' with an additional focus on *'how well'*. In addition, we respond to Storey *et al.*'s invitation for similar studies considering other venues and publication dates by suggesting that considering the state of human factors within different software engineering sub-fields may also be of benefit.

In focusing on APR, our work demonstrates that Storey *et al.*'s findings apply to the specific software engineering area of APR. Whilst APR was not the topic of any of the 151 articles reviewed by Storey *et al.*, our findings largely mirror Storey *et al.*'s that 'although a majority of these papers claim the contained research should benefit human stakeholders, most focus predominantly on technical contributions' and rarely involve human studies [12]. Whilst there is not necessarily any reason to expect that APR would be different to software engineering more generally, we wanted to thoroughly assess the state of human factors research in APR in order to present recommendations specifically tailored to the APR domain. This is important because APR is a new and rapidly advancing field of software engineering – with vast potential to impact upon software developers' work – that has so far not been explored qualitatively or quantitatively to the extent that other areas of software engineering have.

By focusing specifically on APR, we also uncover some strikingly different findings from Storey *et al.*'s study. For example, whilst Storey *et al.* find a high number of papers that use mining software repositories as a research strategy, we find only a small number of APR papers that use software repositories as a source of data, which we discuss in Section 4. This may be a result of the nascent nature of APR, providing an important opportunity to reflect on the state of the field at this early stage. We hope our work may inspire similar studies in more established areas of software engineering to aid comparison. This is important to ensure we gain understanding of different sub-fields, and the similarities and differences between them.

We provide a thorough discussion of human studies within APR in this paper. However, it is worth mentioning here a couple of relevant studies that have been published since we conducted our analysis, and are included in the more recent version of Monperrus's Living Review. Noller *et al.*'s survey of 103 participants found high willingness from participants to review automatically generated patches [21]. The survey results also provide indications of what might increase developer trust in automatically generated patches, such as test cases, explanations of the patch, and evidence of patch correctness. Alarcon *et al.*'s experimental study also considered trust in APR, and found that the source of the repair (human versus automated) had significant influence on trust perceptions and intentions, participants having higher trust in human repairs than automated repairs [22]. Both these studies demonstrate recent advances in human factors research in APR.

## 3 Aims

The aim of this paper is to provide an assessment of the current state of the APR literature when it comes to the consideration of human factors. This paper addresses four core research questions.

- *RQ1:* To what extent, and in what ways, does the APR literature consider human factors?
- *RQ2:* What are the strengths and weaknesses of existing human studies within the APR literature?
- *RQ3:* What are the key findings of existing APR human studies?
- *RQ4:* What future research directions are needed to progress the study of human factors within APR?

In addressing these questions through an in-depth review of 260 Living Review papers, we make the following contributions:

- *Contribution 1:* We provide the first in-depth review of the extent to which human factors are considered in the APR literature.
- *Contribution 2:* Drawing on existing quality assessment criteria for empirical studies in software engineering, we propose a new set of quality assessment criteria. Our criteria aim to be more robust and replicable to allow for future evaluations of human studies in APR. In addition, they are designed for use

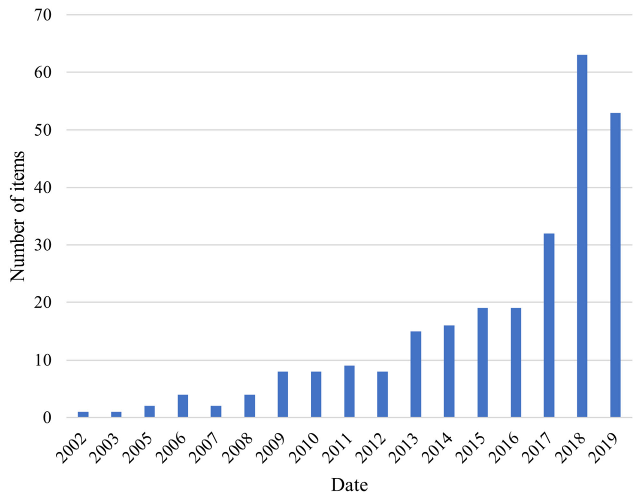

Fig. 1. Publication dates of the living review items (December 2019 version).

with different kinds of empirical study, both qualitative and quantitative.

- *Contribution 3:* We assess APR human studies for quality and identify their current strengths and weaknesses
- *Contribution 4:* We offer a series of recommendations for future APR research that thoroughly considers human factors. In so doing, we hope to inspire a wave of human-centered APR research more attentive to the needs of developers that can subsequently offer APR solutions that meet these needs.

## 4 Methodology

Our review used the Living Review [9] as a corpus and involved two main phases: first, a broad consideration of how much and in what ways APR research is considering developers; and, second, an in-depth quality assessment of human studies within APR. In this Section, we introduce our corpus (the Living Review) and our overall approach, before outlining the main methods we used to answer our research questions. Please see the appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE.2022.3152089, for an overview of each author's involvement.

### 4.1 Our Corpus- the Living Review

We use Martin Monperrus's 'The Living Review on Automated Program Repair' [9] as an already-existing corpus of the key APR literature. The Living Review is a 'live' version of Monperrus's widely-cited bibliography on APR [15] and is updated every two to three months. We used the December 2019 version of the Living Review. Fig. 1 shows the publication dates of all 264 works in the December 2019 version, demonstrating the growth of the APR field since the early 2000s.

Others have previously used the Living Review as a corpus for their own studies. Durieux *et al.* used the Living Review to identify existing test-suite repair tools [23], while Wang *et al.* used it to identify automated patch correctness assessment techniques [24]. Similarly to our own study, Liu *et al.* used the Living Review to extract relevant papers for their critical review of APR evaluation [25]. Using the Living Review as an APR corpus has two other advantages. First, whilst it should not be considered fully comprehensive, by comparison to the other main APR literature review [2], the Living Review includes more than double the number of papers. Second, the Living Review is regularly maintained rather than being static. This means that it would be possible to re-assess our findings in the future, applying our method to new APR work.

TABLE 1
Questions for Review of Living Review Papers

| Question | |
|---|---|
| 1 | Does the paper report on the development of a new APR tool/technique? |
| 2 | Does the paper identify as motivation a problem that developers currently face? |
| 3 | Does the paper state that their APR tool/technique will be helpful to developers (e.g., reduced effort)? |
| 4 | Does the paper state that their proposed APR tool/technique will change developers' activities (e.g., introduce new tasks)? |

### 4.2 Our Approach

As mentioned in Section 2, a review of an under-developed research area, such as human factors in APR, needs a different approach to most systematic literature reviews and research syntheses. As well as exploring trends and themes within the APR literature, we also take a critical approach, identifying the strengths and weaknesses of the current literature in order to make recommendations for APR research that includes consideration of human factors. In taking such an approach, our work can be positioned alongside other recent software engineering publications, such as [26], [27] and [28].

### 4.3 RQ1- To What Extent, and in What Ways, Does the APR Literature Consider Human Factors?

To address RQ1, we conducted a review of the Living Review papers. First, we searched each Living Review paper to identify those that included a human study, using five key search terms: 'user', 'human', 'developer', 'engineer' and 'programmer'. From this, we found 17 items from the Living Review that included a human study (6.5% of the 260 Living Review papers we had access to). We extracted these 17 papers for in-depth analysis in order to answer RQ2. We concentrated our focus for RQ1 on the vast majority of Living Review papers that did not include a human study and whether software developers were nonetheless considered; if so, we explored in what ways.

Similarly to Liu *et al.* [25], we reviewed the 243 Living Review papers with a series of questions designed to critically review an under-explored research domain. These questions (see Table 1) aim to capture the extent to which developers are taken into consideration in APR research.

All 243 Living Review papers (excluding the 17 human studies) were imported into SLuRp, a software system designed for the management of systematic literature reviews [29]. SluRp

TABLE 2
Reviewer Agreement for Each Review Question

| Question | Percent agreement |
|---|---|
| 1 | 91.3 |
| 2 | 72.7 |
| 3 | 77.9 |
| 4 | 82.3 |

automatically highlighted five key search terms (as also used to identify the human studies): 'developer', 'engineer', 'human', 'user' and 'programmer'. We also added a further three key search terms that emerged as useful when trialling the review questions- 'manual', 'expert' and 'practitioner'. These were also highlighted automatically by SluRp. (The number of occurrences of each search term can be found in the Appendix.)

Each paper was reviewed by two authors of this study independently, assigned randomly by SLuRP. The review protocol was: to read the abstract, introduction and conclusion of each paper; and to use the yellow highlighted search terms to review the rest of the paper in order to answer the review questions. One author was excluded from the review process to act as an independent mediator and moderator of disagreements between pairs of reviewers, ensuring consistency across moderation.

The answers to the review questions (see Table 1) were recorded by each reviewer on SLuRp. Each review question had a 'yes'/'no' tick-box answer, followed by free-text space for reviewers to provide cut-and-paste quotations from the paper as evidence for their answer. Quotations were also collected in order to thematically analyse them later, as a way of adding depth to the 'yes' responses.

Three different question versions were trialled with a small subset of eight papers reviewed by all authors. This led to refinement of the questions (with final versions shown in Table 1) and the establishment of shared understanding among the authors. Of the remaining 235 papers that were each reviewed by two authors, there was 79.7% agreement across all of the 940 answers (four 'yes'/'no' questions for each paper). The percentage agreement varied across the four questions, as summarised in Table 2. Cohen's Kappa scores for each pair of reviewers for each question are provided in the appendix, available in the online supplemental material. All findings subsequently reported also include the eight papers used in the trial.

The moderator assessed the 191 conflicts between pairs of reviewers and resolved the majority of these conflicts. Unresolved conflicts (a total of 56, or 6.0% of all answers) were discussed by the pair of reviewers, managed over email in the case of simpler conflicts and verbally for more complex conflicts. All discussion and decisions were noted in SLuRp.

The next phase was to conduct a thematic analysis of the quotations that had been collected for Q2, Q3 and Q4 (see Table 1) as part of the review process. Thematic analysis 'is a method for identifying, analyzing, and reporting patterns (themes) within data' [20] and it is particularly appropriate for more explorative research. Two authors worked together to create a codebook, which included definitions of each code and an example quotation. This codebook was developed inductively (see [30]), working from the data to establish thematic codes, rather than coming up with a predetermined list of themes that were then applied to the data.

Once the codebook was established (see Tables 8, 9 and 10), three authors were assigned on SLuRp to each paper. We assigned three authors, rather than two, because the thematic analysis involved a more complex subjective, interpretive judgement than the yes/no judgement of the review. Multiple thematic codes could be chosen for a single quotation, where applicable. We also included a box for 'other', which could be used for reviewers to suggest additional thematic codes. Again, one author acted as independent moderator of any disagreements, not taking part in the reviewing.

Similarly to the prior process, we undertook pilots to ensure that all authors understood the thematic codes. After

TABLE 3
Thematic Coding Agreement, by Review Question (see Table 1)

| Agreement | Thematic codes for Q2 | Thematic codes for Q3 | Thematic codes for Q4 |
|---|---|---|---|
| Perfect agreement | 18.6% | 12.5% | 8.5% |
| All 3 reviewers agree on at least one code | 41.1% | 26.6% | 17.0% |
| 2 reviewers agree on at least one code | 29.5% | 46.9% | 48.9% |
| No agreement | 10.9% | 14.1% | 25.5% |

TABLE 4
APR Papers That Study Developers Indirectly

| Paper | Nature of study [31] | Findings |
|---|---|---|
| [32] | Bug reports containing automatically generated patches submitted to open source software developers | Bug reports including automatically generated patches were addressed in 66% of cases (over a two week period) |
| [33] | Automatically generated patches submitted as GitHub Pull Requests (PRs) | 5 out of 12 patches were accepted |
| [34] | Automatically generated patches submitted as PRs | 89% of automatically generated PRs were accepted |
| [4] | Acceptance of Getafix fix suggestions at Facebook | 42% of Getafix fix suggestions were accepted |

TABLE 5
Existing Quality Criteria

| Category | Summarised criteria from [31] | Summarised criteria from [35] | Mapping onto our own quality criteria (see Table 6) |
|---|---|---|---|
| 1. Empirical study | N/A | Based on research? | N/A- Prerequisite |
| 2. Research context | N/A | Description of research context? | Aims |
| 3. Research design | Description of experiment design? | Research design appropriate to research aims? | Research design |
| 4. Recruitment | N/A | Recruitment strategy appropriate to research aims? | Recruitment |
| 5. Control group | N/A | Was there a control group? | N/A- Not universal |
| 6. Research units | Description of sample/ experimental units? | N/A | N/A- Not universal |
| 7. Data collection | Description of data collection procedures and measures? | Data addresses research issue? | Data collection |
| 8. Data analysis | Definition of data analysis procedures? | Rigorous data analysis? | Analysis and findings |
| 9. Reflexivity | Discussion of experimenter bias? | Consideration of researcher-participant relationship? | Limitations |
| 10. Limitations | Discussion of study limitations? | N/A | Limitations |
| 11. Findings | Clear statement of findings? | Clear statement of findings? | Analysis and findings |
| 12. Replicability | Evidence that the experiment can be used by others? | N/A | Data collection |
| 13. Research significance | N/A | Study of value for research or practice? | Too subjective |

the full thematic coding was complete, all conflicts were moderated, discussed where needed, and decisions recorded in SLuRp. The agreement levels for the completed thematic analysis are shown in Table 3. To summarise, there was a majority of at least partial agreement for all three questions that were thematically coded: 89.2% for Q2 quotations; 86.0% for Q3 quotations; and 74.4% for Q4 quotations.

Disagreement arose when the quote was slightly ambiguous or required more interpretation. For example, one reviewer commented, ‘partially I would agree with X too, because there is some additional information but it’s not very clear’, whilst another reviewer commented on their interpretation process: ‘I would say that it’s “effort reduction” as I interpret the stated “helpfulness” to be less work to look for where to fix’.

## 4.4 RQ2- What are the Strengths and Weaknesses of Existing Human Studies Within the APR Literature?

To address RQ2, we conducted a quality assessment of the 17 papers from the Living Review that included some form of human study. Human studies were found using the key search terms, outlined under Section 4.3. The criterion for inclusion as a human study was that contact with humans was direct and elicited, what Storey *et al.* refer to as ‘engaging humans’ [12]. We did not include, for example, analysis of developer patches or comments about automatically generated patches that were publicly available on repositories such as Github. However, we did examine all four of these studies that appeared in the Living Review, and their key features and findings are summarised in Table 4.

Of these four studies that did not engage with developers directly, none are large enough to be considered as software repository mining studies, but they share similarities with this approach. Such studies cannot be quality assessed using the same criteria as studies that engage directly with developers to elicit specific data. Developers are not actively participating in a primary empirical study; rather, secondary ad hoc data generated by developers is collected and analysed. Our aim was to establish quality criteria for empirical studies that *directly* engaged humans, and that could be broadly applied to a wide range of user studies; therefore we excluded these four studies from our main analysis. In Section 6, we consider the current paucity of mining software repository studies in APR when compared with Storey *et al.*’s review of software engineering more generally.

We wanted to establish how well direct human studies had been carried out to understand the current state of APR research with developers. We used existing work on quality assessment within SE to consider ways of assessing the APR human studies, specifically two lists of quality criteria that had been developed for use within empirical SE [31], [35]. Table 5 summarises and compares these two lists of criteria by theme.

We used the comparison of these two existing quality criteria to identify six core categories for quality assessment criteria: aims, research design, recruitment, data collection, analysis and findings, and limitations. These were designed to be applicable to all kinds of empirical studies, both quantitative and qualitative. For each of our categories, we developed three separate tick-box criteria, shown in Table 6. Following Dybå and Dingsøyr [35], we chose to use ‘yes’/‘no’ answers. Table 5 demonstrates how our quality assessment criteria map onto the categories we identified in [31] and [35], including why we chose not to use some of these categories. We excluded control group as a quality criterion, because control groups are only appropriate for experimental design not

TABLE 6
APR Human Studies Quality Criteria

| Shorthand | Full criteria |
|---|---|
| Aims | Aim(s) of human study clearly stated |
| Aims-RQs | Research question(s) and/or hypotheses specifically related to the human study |
| Aims-Motivation | Clear motivation for the human study |
| Research Design | Research design clearly described |
| Research Design-Aims | Research design well suited to the aims of the study |
| Research Design-Motivation | Utilised methods clearly motivated |
| Recruitment-Who | Who the participants are and where they were recruited from |
| Recruitment-How | How participants were recruited |
| Recruitment-Motivation | Recruitment choices and/or sampling strategy clearly motivated |
| Data Collection-How | How data was collected is clearly explained |
| Data Collection-What | What kind of data was collected is clearly explained |
| Data Collection- Replicability | Public availability of data collection tools and/or data |
| Analysis and Findings-How | How the data was analysed is clearly explained |
| Analysis and Findings-Motivation | The analytical procedures used are clearly motivated |
| Analysis and Findings-Findings | Findings are clearly described and explained |
| Limitations-Validity | Consideration of validity |
| Limitations-Generalizability | Consideration of generalizability |
| Limitations-Reliability | Consideration of reliability and trustworthiness |

for all empirical studies, and we aimed to develop a list of quality assessment criteria appropriate for all human studies.

Two authors assessed each paper and then discussed their answers. Agreement was often simple, as in the case of one reviewer having missed something. Overall, inter-rater agreement was 78.4%. The Cohen Kappa coefficient was 0.569, indicating 'moderate agreement' [36] and a level of agreement beyond chance. The percentage agreement for each of the quality criteria are provided in the appendix.

Agreement between the authors was generally very strong, except for 'Aims' and 'Data Collection-What'. Disagreement here arose when aims and data collection were reported fairly implicitly, indicating the degree to which core elements of a human study are not necessarily being clearly and explicitly reported.

Having agreed independently in 78.4% of cases, discussion led to quick agreement in a further 20.0% of cases, leaving just 1.6% of cases where the opinion of a third author was needed. Kitchenham *et al.* found discussion a highly important part of quality review processes [37], and this was reaffirmed by our experiences.

## 5 Findings

### 5.1 RQ1- To What Extent, and in What Ways, Does the APR Literature Consider Human Factors?

Table 7 summarises the results of the review questions over the 243 APR papers. These results indicate three core findings. *First, the APR papers make some significant claims about developers without having conducted a human study.* Of the 81.1% of papers that introduced a new tool or technique, 67.0% mention a problem that developers currently face as a motivation for their work and 36.0% specify that their APR tool/technique will be helpful to developers. These statements are made without any interaction with software developers. The evidence base for these claims is discussed in Section 6.

*Second, there remain many papers that do not consider the developer at all and are purely technical in their outlook.* Of the papers that introduced a new tool or technique, 33.0% did not state as motivation a problem currently faced by developers and 64.0% do not specify that their tool or technique will be helpful to developers. These papers are grounded in technical motivations or challenges (such as improving on the shortcomings of other tools or techniques) and the authors' technical achievements and contribution.

*Finally, keeping humans 'out of the loop' would seem to be a key goal of APR*, with only 21.8% of papers that introduced a new tool or technique stating that the tool would introduce a new task or activity for developers, such as verifying and choosing patches or writing a specification. Stressing the lack of required human input was more common.

We considered the quotations collected in more detail using thematic analysis. Tables 8, 9 and 10 show the thematic codes, their definitions, and an indicative quotation from the Living Review papers. Figs. 2, 3 and 4 show the percentage of quotes that were coded with each of the thematic codes (quotations could be coded with multiple thematic codes, where appropriate).

*For Q2 (Does the paper identify as motivation a problem that developers currently face?), the main motivation identified in the papers was that fixing faults takes a lot of developer time and effort*, featuring as a motivation for over 60% of the quotations collected for Q2. The next most prevalent theme was

TABLE 7
Review Question Results

| Question | Percent of papers marked 'yes' |
|---|---|
| 1-Does the paper report on the development of a new APR tool/technique? | 81.1 |
| 2-Does the paper identify as motivation a problem that developers currently face? | 54.3 |
| 3-Does the paper state that their tool will be helpful to developers (e.g., reduced effort)? | 29.2 |
| 4-Does the paper state that their tool/technique changes developers activities (e.g., introduces new tasks)? | 17.7 |

TABLE 8
Thematic Codes for Q2- Does the Paper Identify as Motivation a Problem That Developers Currently Face?

| Thematic code | Definition | Indicative quotation |
|---|---|---|
| Difficulty | Developers find program repair hard | Manually debugging a defective program is notoriously difficult |
| Fault localisation | Localising the fault is a challenge for developers | The programmer's ever recommencing fight against error involves two tasks: finding faults; and correcting them |
| Frustration | Developers find program repair frustrating | Even a non-scalable automatic repair method can help save a lot of time and avoid much frustration |
| Mistakes when coding | Developers prone to making mistakes in original code | Programmers make mistakes |
| Mistakes: when fixing faults | Developers prone to making mistakes when fixing faults | Human developers often introduce new defects over the course of repairing others |
| Need for advanced skills | Program repair requires the developer to have advanced skills | Unfortunately, correctly fixing distributed timing bugs is challenging for developers, as it involves global reasoning beyond one thread or one node, and often requires non-traditional synchronization |
| Shortcomings of a current repair technique: too many options | Current technique provides developers with too many options to sift through | Even with automated bug localization, the programmer must still assess these locations to choose where and how to fix the program |
| Shortcomings of a current repair technique: understandability | Current technique yields output that developers struggle to understand | This kind of correction is not readable and cannot be easily understood and verified by the design engineer |
| Tedium | Developers find program repair boring | Even if the bug's cause is known, detecting a bug in such programs and generating a bug fix patch manually is a tedious task |
| Time/effort | Program repair takes a lot of developer time/effort | Localizing and fixing bugs is known to be an effort-prone and time-consuming task for software developers |
| Time pressures | Program repair has to be conducted within pressurised timeframes | Modern software applications must satisfy strict release requirements that impose short bug fixing and maintenance cycles, putting significant pressure on developers |

'difficulty', that developers find program repair difficult. This applied to over 30% of Q2 quotations.

*For Q3 (Does the paper state that their tool will be helpful to developers (e.g., reduced effort)?), the key finding was that reduction of developer effort was the main benefit promised to developers*. Perhaps surprisingly, the theme 'usability' was the least common, with few papers claiming that they had developed a tool that would be usable by developers. As none of the papers in this part of our review included a human study, none of these assertions are based on having trialled or evaluated the tool or technique with developers.

*For Q4 (Does the paper state that their tool/technique changes developers' activities (e.g., introduces new tasks)?), the most common additional task required was that developers had to provide some form of specification to the APR tool*. Whilst most of the codes are a type of additional task, the codes 'added value' and 'flexibility' refer to how this additional work was positioned in the papers. 'Added value' refers to when the paper stressed that developers were given information to assist them with the new task and 'flexibility' refers to when the paper indicated that the additional tasks gave developers choice or options over the process. Flexibility was the more prevalent of these themes, representing the main way in which the introduction of new tasks was positioned, and perhaps justified, emphasising developer choice and agency.

*Another key finding for Q4 is that, whilst manual debugging was presented as difficult (Q2 quotations), the new developer tasks resulting from APR tools/techniques were presented as straightforward or easy*; for example, 'the programmer is required to define a catalog of hotspots, syntactic constructs considered to be error-prone. Instead of manually searching for hotspots, programmers *just* define a catalog of syntactic constructs' [38] (our italics). It is unclear to what extent such tasks are indeed easy.

To summarise, in answer to 'RQ1- To what extent, and in what ways, does the APR literature consider human factor?', we find that around two thirds of APR papers make some significant claims about developers without having conducted a human study, while around a third of papers are purely technical in outlook not considering human factors at all. A key stated motivation for APR research was that developers spend much time and effort fixing faults, while a main stated benefit of the research was to reduce developer effort. Additional tasks required as a result of APR, such as providing specifications, were generally positioned as straightforward.

### 5.2 RQ2- What are the Strengths and Weaknesses of Existing Human Studies Within the APR Literature?

We found 17 human studies in the Living Review, including one PhD thesis [39]. Date of publication ranged from 2006 to 2019, with only three studies published between 2006 and 2013 and six studies published in 2018. This does suggest that the number of human studies is increasing as the field

TABLE 9
Thematic Codes for Q3- Does the Paper State That Their Tool Will be Helpful to Developers (e.g., Reduced Effort)?

| Thematic code | Definition | Indicative quotation |
|---|---|---|
| Difficulty reduction | Proposed tool/technique removes a challenging task | Beanbag program is much easier than manually implementing the fixing procedure |
| Efficiency | Proposed tool/technique makes things more efficient | Wolverine, by virtue of this seamless integration of debugging and repair, allows advanced debug-repair strategies [. . .] facilitating significant speed ups during repair |
| Time/effort reduction | Proposed tool/technique will save developer time/effort | By automatically building fixes for bugs in real-world programs, it can help curb the large amount of resources - in time and effort – that programmers devote to debugging |
| Usability | Proposed tool/technique is easy to use | This approach [provides] a clear, predictable fixing semantics, so that end users can clearly know how their updates affect other parts of the model |
| User understanding | Proposed tool/technique contributes to user understanding | Programmers viewing the analysis output can use such patches as guides, starting points, or as an additional way of understanding what went wrong where |

grows, though only in proportion to the growth of the field (as 2018 has the largest number of papers in the Living Review, see Fig. 1).

Table 11 highlights some key details of the human studies we found in the Living Review: the type of study, the number of participants and, where it was stated, who these participants were. UGs refers to undergraduate students; PGs to postgraduates; and AMTs to Amazon Mechanical Turk workers. Table 11 shows that *the most commonly used research methods were (quasi)experimental and survey-based*. The number of participants, however, varied greatly, ranging from just one to several hundred. The implications of the common types of study and the kinds of participants recruited are discussed in Section 6.

*APR Human Studies are Highly Tool Specific*. Fourteen of the seventeen papers introduce a new tool or technique that the authors developed and then conduct a human study to test out this tool/technique. The exceptions were [39], [43] and [53]: these all presented participants with patches generated from other tools or taken from benchmark suites. For example, Fry *et al.* consider how maintainable participants considered automatically generated patches taken from a benchmark suite, while Tao *et al.* study developers' reactions to high- and low-quality patches. The implications of this are also considered in Section 6.

### 5.2.1 Quality Assessment Results

We evaluated each of the 17 human studies according to the 18 quality criteria shown in Table 6, meaning that each human study was given a score out of 18. The highest score achieved by a human study paper was 17 [53], while the lowest score was 2. The mean score was 8.3 and the median score was 9, demonstrating the fairly low scores achieved by individual papers.

The quality criteria also helped to highlight the strengths and weaknesses of human studies in the APR field, summarised in Fig. 5. This figure highlights some major weaknesses

TABLE 10
Thematic Codes for Q4- Does the Paper State That Their Tool/Technique Changes Developers' Activities (e.g., Introduces new Tasks)?

| Thematic code | Definition | Indicative quotation |
|---|---|---|
| Added value | The developer is given more information than before | The fixes that pass validation are presented to the user, heuristically ranked according to how likely they are correct |
| Additional tasks required: manual application | The developer has to apply the repair | By suggesting that an identifier be inserted or modified, the choice is still up to the programmer |
| Additional tasks required: testing | The developer has to carry out tests | We assume that programmers should decide the behavior of a merged program and usually prepare test cases to check if it behaves correctly |
| Additional tasks required: spec | The developer has to provide some sort of specification | To fix the bugs in a program with this novel approach, a user needs only to provide either a formal specification or a set of unit tests |
| Additional tasks required: verification | The developer has to check provided fix and/or choose between possible fixes | In general, user interaction is necessary to select the desired repair from the set of possible repairs given by the algorithm |
| Flexibility | The developer is given choices/ options on the process | In practice, the choice of which mutant operators to use would come down to the intuition, needs, and resources of the programmers and software engineers applying the strategy |

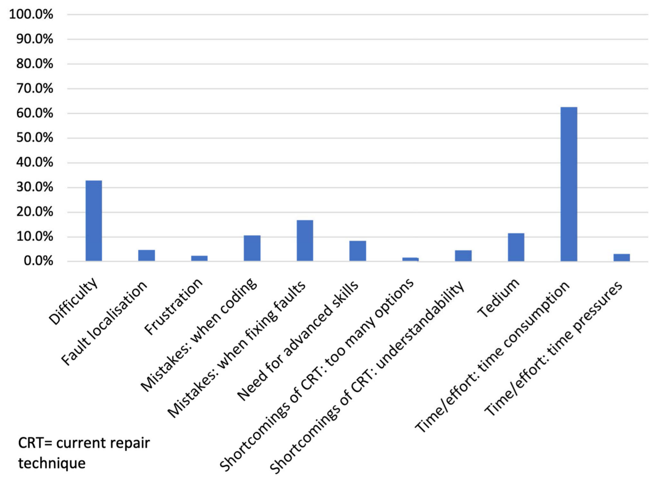

Fig. 2. Thematic code percentages for Q2 quotations - Does the paper identify as motivation a problem that developers currently face?.

in the current APR human studies. *Specifically, APR human studies are not currently well motivated and justified.* Only one paper explained why the method they used had been adopted and was appropriate: Cambronero *et al.* explained that their experiment had been chosen to 'model a scenario, inherent in the use of generate-and-validate automatic patch generation, in which the developer is given patches that validate but may or may not be correct' [41]. The remaining studies gave no justification for their choice of method and why it might be well suited to their research aims and questions: in such cases, the method was just announced without any motivation. Similarly, analytical procedures taken - such as the choice of particular statistical tests - were rarely justified. Few papers explained why they had recruited the participants they had, or what their sampling strategy was and why this was appropriate. All the papers seemed to use some form of convenience or volunteer sample, but the nature of the sample was never identified or explained. Though a little more common, very few papers explained why their chosen participants were the right people to take part in the human study. Based on this overall absence of thorough explanation and justification, the current state of APR human studies seems to be fairly under-developed.

A lack of maturity in current APR human studies is also demonstrated by examples where the conclusions offered by studies were not in line with the data presented. One study

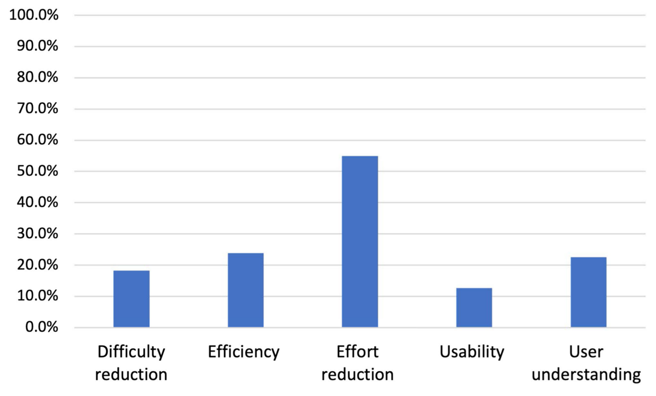

Fig. 3. Thematic code percentages for Q3 quotations - Does the paper state that their tool will be helpful to developers (e.g., reduced effort)?.

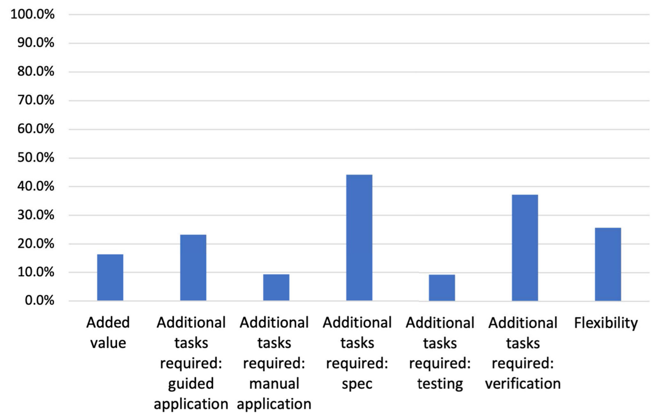

Fig. 4. Thematic code percentages for Q4 quotations - Does the paper state that their tool/technique changes developers activities (e.g., introduces new tasks)?.

concluded that 'through a user study, we have shown that Fix-It can reduce the human effort in code review to a significant extent' [40]. However, the user study included only one person, who considered that Fix-It was able to provide fixes in 43% of cases, which does not necessarily equate with a reduction in human effort (as it does not take into account time taken to validate fixes, etc.). Similarly, another paper, reporting on participants' answers to the question 'does FIXML help to understand your mistakes when you cannot resolve them by yourself?', includes 'neutral' responses with 'yes' responses and as a result over-states how many participants gained understanding through using the tool [49]. This also indicates that peer reviewers may not give due consideration to human studies that represent just a small part of predominantly technical papers, meaning that misleading reports of human study findings may be published.

In answer to 'RQ2- What are the strengths and weaknesses of existing human studies within the APR literature?', we find that human studies within APR are currently characterised by a lack of methodological diversity. Whilst the human studies generally had well described research design, data collection and findings, they were often poorly motivated with little justification of key decisions (such as choice of methods or type of participants).

## 5.3 RQ3- What are the Key Findings of Existing APR Human Studies?

We also provide here a summary of the key findings reported in the human studies. The findings can be placed into three main categories:

- *Attitudinal:* Participants' reactions to, and feelings about, the tool
- *Tool performance:* How the tool performed in the context of the human study
- *Human performance:* How participants performed under the task conditions

Table 12 summarises the key findings of these human studies, split into these key categories and divided into subcategories.

*The findings in the 'human performance' category (i.e., how participants performed under experimental conditions) are surprisingly mixed.* Two papers found that having access to the tool or its generated patches did not improve participants' ability

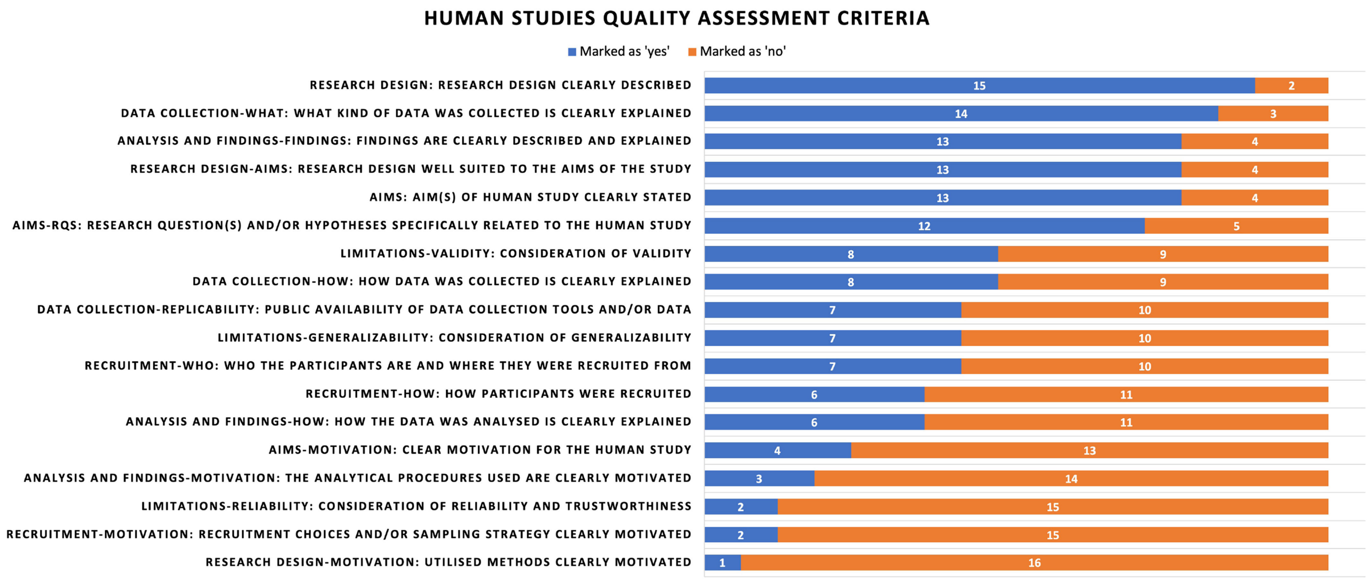

Fig. 5. Human studies strengths and weaknesses.

to successfully complete the task [41], [47], whilst two papers found that it did [46], [53]. Similarly, two papers reported that having access to the tool or its generated patches did not improve participants' ability to complete the task quickly [41], [53], whilst two papers found that it did [46], [47]. One study found that one group of participants (teaching assistants) completed the task quicker with access to the tool and one group (students) completed it slower [55].

Not all the findings of these human studies were positive. In contrast to the claims made in Living Review papers *without* human studies that APR tools would be helpful to developers, several of the human studies did not show that the APR tools were in fact helpful. Cambronero *et al.*'s study found little difference between the performance of two groups of participants in terms of time taken to perform the tasks and the number of correct patches submitted [41]. Both groups had been asked to repair defects and were given the location of the defective lines of code, but one group had access to five automatically generated patches, of which one was correct. Cambronero *et al.* concluded that 'solely providing subjects with automatically generated patches may not be sufficient to see an effect in terms of patch integration productivity [...] Subjects spent most of their time trying to understand the defect and the way the provided patches related to the original source code containing the defect' [41]. From this, it seems that an understanding of the original defect may be a prerequisite for developer patch acceptance, indicating that work needs to be done into how automatically generated patches are presented to developers.

Daniel *et al.*'s study found that more faults were introduced by participants that had access to their tool than those that did not. In contrast to the caution exhibited by Cambronero *et al.*'s participants regarding patch acceptance, Daniel *et al.* suggest that these participants might have 'become overly reliant on the tool. This can be mitigated by training users to carefully inspect the repairs suggested by ReAssert rather than accepting them blindly' [42]. The results from [41] and [42] present a more complex picture

TABLE 11
APR Human Studies Included in the Living Review

| Paper | Experiment? | Survey? | Other? | Number and make-up of participants |
|---|---|---|---|---|
| Balachandran [40] | | | Elicited developer feedback | 1 |
| Cambronero [41] | X | X | | 12 (all PGs) |
| Daniel [42] | X | X | | 18 (13 PGs, 3 UGs, 2 industry professionals) |
| Fry [43] | X | X | | 150 (27 UGs, 14 PGs, 116 AMTs) |
| Gulwani [44] | X | X | | 52 |
| Hata [45] | | X | | 20 (5 UGs, 14 PGs, 1 Professor) |
| Kaleeswaran [46] | X | | | 10 (2 PGs, 8 industry professionals) |
| Kalyanpur [47] | X | | | 12 |
| Kim [48] | | X | | 153 (17 PGs,72 UGs, 164 industry professionals) |
| Le [39] | X | X | | 35 |
| Lee [49] | X | X | | 18 (all UGs) |
| Liu [50] | | X | | 7 |
| Mahajan [51] | | X | | 240 (all AMTs) |
| Mahajan [52] | | X | | 37 (all UGs) |
| Tao [53] | X | X | | 95 (44 PGs, 28 industry professionals, 23 AMTs) |
| Tomida [54] | | | Tool demo followed by discussion | 7 (4 PGs, 1 UG, 2 professors) |
| Yi [55] | X | X | | 300 (263 UGs and 37 TAs) |

TABLE 12
Key Findings of APR Human Studies

| Attitudinal | No. of human studies |
|---|---|
| Participants found the tool/patches/repairs useful | 5 [42], [44], [45], [46], [49] |
| Participants expressed some critical feedback about the tool/patches/repairs | 3 [42], [50], [53] |
| Participants rated after-repair version of website better than pre-repair version (readability/aesthetics) | 2 [51], [52] |
| Participants considered the tool able to fix errors | 1 [40] |
| Participants felt that the tool/patches/repairs would save their time | 1 [50] |
| Participants felt that the tool/patches/repairs would lead them to respond quicker to a bug report | 1 [50] |
| Participants rated the tool's generated patches better than those generated by another tool | 1 [48] |
| Participants rated the tool's generated patches similarly to human-written ones | 1 [48] |
| Participants considered the task(s) easier when they had access to the tool/patches/repairs | 1 [46] |
| **Tool performance** | **No. of human studies** |
| The tool was successfully able to fix errors and/or generate patches | 3 [42], [44], [49] |
| Generated patches/repairs were similar to human-written ones | 1 [42] |
| **Human performance** | **No. of human studies** |
| Access to tool/patches/repairs decreased the time it took participants to debug/complete task | 3 [46], [47], [55] |
| Access to tool/patches/repairs did **not** decrease the time it took participants to debug/complete task | 3 [41], [53], [55] |
| Different types of participants made varyingly effective use of repairs/patches | 2 [53], [55] |
| Access to tool/patches/repairs improved participants' ability to debug/complete task | 2 [46], [53] |
| Access to tool/patches/repairs did **not** improve participants' ability to debug/complete task | 2 [41], [47] |
| Access to tool/patches/repairs led to participants introducing more faults that those who did not have access | 1 [42] |
| Participants were able to understand generated patches with documentation as well as they did human patches | 1 [43] |

than the simplistic message of 'our new tool will help software developers' provided by many of the Living Review papers that do not include a human study. In particular, the contrasting findings relating to patch acceptance suggest that the issue of what influences developers' patch acceptance requires further research. Daniel *et al.*'s suggestion of the risk of developer complacency also opens up key research questions around how patch acceptance is best managed and what tooling might be appropriate to ensure careful developer validation of patches.

Alongside positive feedback, there was also critical feedback provided by participants in some of the human studies. In Tao *et al.*'s study, participants said: that the generated patches may be confusing, misleading or incomplete; that they may either overcomplicate or oversimplify the problem; that they may not be helpful for unfamiliar code; that they may not work if the test suite is not sufficient [53]. The participants in Liu *et al.*'s study voiced some concern about the accuracy of the patches generated by the tools [50]. Daniel *et al.*'s participants expressed the concern that the tool could make developers more careless (and this seems to be supported by the fact that participants with access to the tool introduced more faults) [42]. Again, this complicates the more simplistic picture provided by the papers reviewed in the first part of our study. Research is needed to explore these developer concerns in more depth and identify ways of mitigating them through tool design.

To summarise, in answer to 'RQ3- What are the key findings of existing APR human studies?', we find some mixed findings as to the helpfulness of APR for participants (in terms of aiding participants in their completion of a given task, for example), as well as evidence of negative developer feedback.

## 6 Discussion

### 6.1 Fault Fixing as 'Difficult': Common Knowledge?

We found little empirical evidence presented for fault fixing being time-consuming and/or difficult. Analysis of the Q2 quotes (*Does the paper identify as motivation a problem that developers currently fix?)* demonstrates that the idea of fault fixing as time-consuming or difficult was often expressed in general, rather than specific terms. Of the 40 quotations that were coded with 'difficulty', 15 referred to difficulty in very generic terms. The same applies for 26 out of 75 quotations thematically coded with 'time/effort'. In addition, the difficulty and time-consuming nature of fault fixing is often referred to as established fact or as 'common knowledge', and rarely backed-up with an appropriate citation. Only one of the references used to support the idea of fixing faults being time-consuming and/or difficult was in fact an empirical study with developers. For the code 'difficulty', over half the quotations did not include a reference at all to support this claim.

The notion that fixing faults is a difficult activity is not necessarily backed up by the literature. As recently as 2018, Beller *et al.* stated: 'surprisingly 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' [56]. Böhme *et al.* agree, asserting that 'how humans actually debug is still not really well explored' [57]. Such work suggests that we don't yet know much about how developers fix faults, let alone how they feel about it.

### 6.2 Do Developers Want to be Out of the Loop?

One reason for papers not often stating that their proposed APR tool or technique would lead to new tasks or activities for developers is that full automation of repair is a key goal of many papers. Full automation may be a key technical goal in terms of proving the efficacy of an APR system, but there is a need to explore whether developers in fact *want* to be out of the loop in this way. Some findings from existing human studies would suggest that developers do not necessarily want this, such as the participants in [41] who took a long time trying to understand both the original defect and the automatically generated patch. This demands further research, otherwise we may find APR innovation that is highly unpopular with developers and at odds with their workplace values.

The concern expressed by participants in [42] that developers might become careless also points to the potential advantages of manual debugging, such as taking care when writing code to avoid needless errors. Manual debugging also gives developers the opportunities to learn from their mistakes, and future work needs to consider the risks of deskilling posed by APR advances.

The idea of program repair bots is an interesting one in terms of allowing for interaction between the human developer and the APR tool. Monperrus, for example, envisages 'conversational systems for patch explanation: developers would be able to ask questions about the patch behaviour, and the program repair bots would answer to those questions' [58]. Van Tonder and Le Goues also consider the possibilities of repair bots, the effectiveness of which depend 'on successful integration with human processes of software development' [59]. Whilst this work is currently nascent and at the conceptual stage, our research findings suggest repair bots could be a promising direction, particularly, for example, for aiding developer understanding of automatically generated patches.

## 6.3 Beyond Tool-Specific Studies

APR human studies are currently highly tool-specific, evaluating in some way an APR tool or technique introduced in the rest of the paper. This is significant, because these human studies reveal little about how developers feel about APR more generally. This may indicate an assumption that APR is the future and will be easily embedded into industry, and it is just a question of the refining of specific tools or techniques, or even a competition to produce the best. Whilst developers may comment on the likelihood of adopting the specific tool under investigation, this does not necessarily reveal much about their attitudes towards APR tools and techniques more generally, often instead reflecting the particular advantages or disadvantages of the proposed tool. Owing to this specificity, the human studies may not provide other researchers working within the APR field with many takeaways for their own work, the findings instead being more applicable to how the proposed tool/technique might be further developed.

It is also challenging to compare the results of these human studies. Whilst one tool-specific study did compare their own proposed tool to GenProg [48], a genetic-programming based repair tool [60], the rest of the studies compare their proposed tool to 'no tool'. In addition, the studies conducted – though largely of similar types, a survey or a task-based experiment – varied. For experimental studies, the type of tasks performed by participants differed, as did the control and experimental conditions. These conditions included, for example, one group of participants with and one group without the repair technique/tool introduced in the paper [42], [47], [55] or one task where participants had use of the tool and one task where they could *not* use the tool [46]. The different experimental conditions make comparison of the tools challenging.

The surveys also had highly varied approaches. Two survey-based studies by the same authors elicited feedback on the aesthetics of an original webpage versus the same webpage repaired by their tool [51], [52]; one involved asking a developer to feedback on the tool's capability to find fixes for different faults [40]; one involved a small number of developers being asked whether the patches generated would save time and be beneficial [50]; one asked participants to review patches [48]; and the remaining study asked participants to rank fixes (both incorrect and correct) according to their understandability and helpfulness [45]. Again, this variety of empirical measures limits comparability between human studies when it comes to considering the usefulness of these APR tools for developers. This makes it difficult to understand what kind of APR tools might most benefit developers. In addition, fewer than half the human studies included some form of replication package, meaning that it would be difficult to replicate many of the studies with a different tool/technique to enable comparison.

## 6.4 The Limits of Usability

The majority of human studies within APR share a common emphasis — i.e., the use of human participants to test or validate a proposed tool or technique. Whether stated or not, all these studies test the hypothesis that their proposed tool or technique leads humans to achieve a better outcome in some way (be that faster, more efficient, or more correct). Whilst there might be some element of attitudinal study, such as participants' opinions on the helpfulness of the tool, the main focus is very much on the tool or technique's usability, with human studies used to validate the tool, and confirm its efficacy.

However, there are limits to the focus on usability. Singer *et al.* highlight the strong focus within tool development on usability testing suggesting that usability does not in fact equate with usefulness, and argue that 'the usability approach cannot speak to the issue of whether a user will adopt and use a new tool in the workplace because that is not the point, or the focus of usability' [61]. This is because usability testing normally occurs outside normal work settings, this being the case in the APR human studies we evaluate. Experimental studies separate the user from their normal forms of behaviour as they are isolated from the other resources they might usually use in their work, such as in-house tools or the advice of colleagues. Singer *et al.* highlight that 'during usability testing, the user is essentially forced to use the software. In consequence, it is impossible to collect data on whether the user would use the software if he [sic] were given a choice between his [sic] existing work practices and the new software' [61]. This highlights the need to ground new tool design in the study of both existing practices and the use of tools in realistic settings.

### 6.5 Participation of Professional Developers

Professional developers were under-represented as participants in the APR human studies. Of the seventeen studies, 11 involved a majority of either undergraduate or postgraduate students [41], [42], [44], [45], [49], [52], [54], [55], or Amazon Mechanical Turk workers [51], or both [43], [53] (see Table 11).

The use of students in software engineering research has been recently defended [62], but it remains a contested area [63]. As discussed earlier, we found that very few papers provided a justification for their recruitment choices, i.e., why students might be appropriate participants. Given the lack of rationale provided, it is therefore hard to judge whether students were suitable research participants for the APR human studies. However, we would argue that the under-representation of professional developers in APR human studies is problematic for a field that is highly motivated, as shown in the first part of our study, by problems faced by developers.

### 6.6 Paucity of Mining Software Repository Studies

Storey *et al.* [12] include mining software repository studies in a category they term 'data strategies', defined as 'empirical studies that rely primarily on archival, generated or simulated data', and find that this strategy is the most commonly used in their sample of papers. By contrast, we find only a small number of papers in our study of APR publications that engage with data from software repositories. This probably reflects the nascent stage of APR.

Should APR aim to mirror other fields in software engineering and increase its number of software repository mining studies, such as submitting patches and collecting acceptance rates and feedback? Certainly, such studies are highly scalable with the potential to yield large quantities of data, as well as often being straightforward to replicate. They may also offer a high degree of realism. However, these studies also have their limitations. As Weimer comments about his own study [32], 'this experiment only shows that patches work, it does not show why' and that 'probing studies, presumably involving human subjects, remain as future work'. Whilst we welcome the insight that might be provided from larger studies gathering developer feedback more indirectly, we argue this should complement, rather than replace, more in-depth empirical studies with developers. Storey *et al.* also caution against over-reliance on data mining studies, specifying that the data collected is 'inadequate for understanding previous human behaviour and can be misleading in terms of predicting future developer behaviours' [12].

### 6.7 Barriers to Conducting Human Studies

It is not clear *why* the quality of human studies within APR is not higher. It may be that APR researchers tend to be technical experts who lack the skills and experience to perform high quality human studies (as also speculated by Storey *et al.* [12]). Or it may be that APR researchers struggle with the barriers to user evaluation in SE that Buse *et al.* report, such as difficulty recruiting participants and the time needed to design and carry out human studies [14]. However, it should be noted that these barriers are not necessarily any different in software engineering than in other disciplines where user studies are much more readily carried out, such as psychology. Buse *et al.* also suggest that peer reviewers tend to look for some form of user evaluation, which may have the unintended side effect of small user studies being added to predominantly technical papers. This suggests that applying guidelines to peer review of user studies would be helpful and such guidelines exist [64]. However, future work is needed to establish the reasons underpinning the lack of quality in human APR studies that we report.

## 7 Recommendations: RQ4- What Future Research Directions are Needed to Progress the Study of Human Factors Within APR?

In response to our analysis of the APR literature and to answer RQ4, we make a series of recommendations for improving the study of human factors within APR. These recommendations are split into three key areas: methods; reporting of human studies; and the scope of human factor research. Both 'methods' and 'reporting' largely respond to the weaknesses of APR human studies that we uncovered through our quality assessment. In the 'scope' recommendation, we consider the fact that the APR human studies we considered were mainly tool specific and suggest that APR human studies could be broadened in scope to include more general studies as well as tool evaluations.

It should be noted here that we are not recommending that *all* APR research include a human study, but rather that *more* human studies be conducted in APR and that attention is given to their quality. Particularly, we would encourage studies that give greater attention to what the developer needs and wants from APR, as well as well-conducted evaluation studies using professional developers with varying levels of experience of APR technology (from none to expert). Kirbas *et al.*'s previous work on the adoption of APR at Bloomberg, London [65], reports scepticism and suspicion of APR. Only through exposing developers to the technology and its benefits did developers begin to view APR as a useful and positive influence in their roles. Work on fine-tuning and improving the human experience of APR tools in Bloomberg is an ongoing challenge and not static.

Whilst we consider APR to be distinct in its potential impact upon software developers, it is not necessarily the case that new methods or techniques are required to research its human factors. As such, the recommendations we offer here may well be applicable to other software engineering domains.

### 7.1 Methods

*Human Studies Should Make use of a Diversity of Methods.* The small number of existing human studies demonstrate a reliance on very few methods, being either some form of controlled, task-based experiment or a survey. Both of these methods have distinct advantages. A controlled experiment, for example, allows the researcher to consider the impact of a single variable (in this case, use or non-use of the proposed APR tool or its generated patches) in a controlled environment that reduces the potential impact of other variables. A survey allows the researcher to collect the views and opinions of more participants than could feasibly take part in a controlled experiment and enables the collection of both qualitative and quantitative data. However, surveys and controlled experiments, like all methods, have their limitations.

The study of human factors in APR would benefit from far greater methodological diversity of a relatively radical nature. For example, we see few industry-academic studies where the authors have actually spent concerted amounts of time *in the organisation* they've collaborated with. Seminal and ground-breaking work on bug detection and prediction by Ostrand, Weyuker and Bell in the past [66] has shown the value of joint understanding, knowledge and co-operation between industry and academia. Both Ostrand and Weyuker worked in academia and industry during periods of their research. APR requires understanding developer processes, nuances and/or tacit knowledge that cannot be gained through on-site research meetings only. Whilst valuable approaches, some of the most commonly-used methods in APR research currently (e.g., surveys or experiments) may not map onto the use of planned APR tools or techniques in natural and realistic settings. Ethnographic studies offer one possible approach for studying how a tool or technique might be used in 'messier', but more realistic, environments; see [67] for an overview of using ethnography in software engineering research.

### 7.2 Reporting of Human Studies

*Human Studies Must be Comprehensively Reported.* One perhaps unintended consequence of the increased focus on human factors in software engineering is minimal human studies being tacked onto the ends of predominantly technical papers. In the majority of cases, these very small studies were poorly justified, contextualised and designed. There is no doubt that the technical and human need to be more fully integrated into software engineering publications, reflecting the inherent socio-technical nature of the software engineering enterprise. However, a scantly described study taking up a couple of a paragraphs in a full paper does not strike us as the right approach. As predominantly technical papers are likely to be reviewed by academics with predominantly-technical expertise and less expertise in human studies, poorly-designed human studies that make up a very small part of the paper may escape the attention of peer reviewers. We found several examples of human studies where results were erroneously reported (see 5.3). However, if such findings were taken at face value they could negatively impact upon future work. Human studies should be thorough and they should also be fully reported in order to allow for replication, where necessary.

*The Need for Clear Motivation.* The human studies were most clearly lacking in terms of motivation and provision of rationale. This included motivation for performing the human study at all and clearly motivated research methods and research design, recruitment and sampling rationale, and data analysis approaches. Justification of key methodological decisions and choices should be a cornerstone of all human studies, and be taken into account in peer review processes.

### 7.3 The Scope of Human Factors Research

*Human Studies Should be General as Well as Tool-Specific.* The human studies reported were predominantly tool-specific; that is, they reported on the results of a study of a particular tool or technique. This is an important way in which academic software engineers test and validate the APR tools or techniques they are developing. However, there is also a need for some more general studies of APR to address rather different kinds of research questions. For example, how do software practitioners' currently find and fix faults? Debugging, and its associated processes, is an under-researched and poorly understood area [68], yet understanding these processes plays a pivotal role in considering how APR is best introduced to developers.

Other questions include: How do developers feel about proposed APR tools and techniques? How do they think APR would change their workflow and the everyday nature of their working life? What disadvantages do developers perceive in APR? The work in [65] showed that an unanticipated side-effect of APR was greater awareness of opportunities by developers for cleaning and refactoring their code [69], in essence widening the reach of APR.

All these types of questions would allow for the development of increased understanding of software practitioners' attitudes towards APR and most importantly, if negative, how to change those attitudes. Whilst there are notable exceptions (such as Facebook's adoption of Getafix), adoption of academia-built tools within industry remains uneven and slow [70], [71]; and understanding software developers' attitudes towards APR might help diminish this industry-academia gap. It is also important to understand the gaps between academic and industry priorities for the adoption of APR, as a recent paper on the introduction of APR at Bloomberg demonstrates [65].

## 8 Threats to Validity

In this Section, we highlight and discuss two key threats to validity: the selection of papers we reviewed; and the interpretive judgements involved in reviewing.

Rather than carrying out our own systematic literature review, we used an existing corpus of APR literature, the Living Review [9]. The Living Review is a large and comprehensive bibliography, and has been used by other research as a corpus [16], [23], [24]. However, it may not be fully comprehensive as it is maintained by only one person, and in addition it may be biased towards more technical studies at the expense of more developer-focused research. To mitigate this threat, we carried out a search for 'automatic program repair' AND ('human factors' OR 'human study' OR 'user study') on ACM Digital Library, IEEE Xplore and arXiv. We looked at all the papers that emerged from these searches and, by using the protocol described in Section 4 to determine whether a paper included a human study, found one APR human study that was not included in the Living Review. This human study [72] confirms our core findings in that it is a minimally-described experiment (with participants asked to review human-written repairs and automated repairs), featuring undergraduate students and Amazon Mechanical Turk workers as its participants. As a result, we are confident that our quality assessment of human factors within APR is representative of the APR literature.

All parts of our review involved human judgement and interpretation, which poses challenges for replicability. To address this threat, all reviewing involved at least two authors. Where the judgement involved was more complex (as in the case of the thematic analysis), three authors were allocated to each review. All conflicts were thoroughly discussed, and discussion was recorded in SLuRp.

It should be noted that our findings are not generalisable to other areas of software engineering, and future research would be needed to explore the state of human factors research in other domains. However, our method (such as the questions used to answer RQ1 and our quality criteria) could be widely used within other SE domains.

## 9 Conclusion

This paper has presented the first review of the state of human factors research in APR. In answer to RQ1 *To what extent, and in what ways, does the APR literature consider human factors?*, we find that developers are very often talked *about* but far less frequently talked *with*. In addition, many of the claims made about developers are unsubstantiated and presented as 'common knowledge' with little evidence to back them up. In fact, we know very little about developers' experiences of debugging [56], [57], let alone their feelings about this part of their work. Of the papers that introduced a new tool or technique, there were also just over one third that were not motivated by a problem currently faced by developers and were purely technical in outlook.

RQ2 assessed the strengths and weaknesses of existing APR human studies. We find that human studies within APR are rare (less than 7% of papers in the Living Review) and of highly mixed quality. In particular, we find that human studies in APR are not sufficiently and clearly motivated, with little rationale provided for key decisions made, such as research design and participant recruitment strategies. We also find that APR human studies are highly tool-specific, telling us little about how tools compare to each other and even less about how developers might feel about APR more generally. There has also been as of yet little research with professional developers, many studies recruiting undergraduate students and/or Amazon Mechanical Turk workers as participants.

The findings of the existing APR human studies (*RQ3: What are the key findings of existing APR human studies?*) reveal a mixed picture in terms of the efficacy of APR tools and techniques when it comes to their actual use, prompting questions regarding how automatically generated patches are best presented to developers. More research is needed to thoroughly investigate developer concerns related to APR and consider how to carefully address these concerns. We also find that APR human studies are highly tool-specific, telling us little about how tools compare to each other and even less about how developers might feel about APR more generally.

In our recommendations (*RQ4- What future research directions are needed to progress the study of human factors within APR?*), we suggest that future APR research should be not only tool-specific but also broader in focus, considering developer attitudes towards APR more generally, as well as exploring how APR tools would fit into developers' existing workflows, complementing rather than disrupting existing practices. We also recommend that future APR research make use of a diversity of methods, including qualitative research methods, and that, in publications, human studies be comprehensively reported and clearly motivated.

There are many thorough and sophisticated human studies in software engineering (for example, [73], [74]). There is no need for APR researchers to re-invent the wheel when it comes to empirical studies, but collaboration with software engineering researchers more focused on the human-side or with social scientists is clearly needed. We suggest that interdisciplinary research that brings together both social and technical expertise may be an important future step in APR.

Our key finding that human factors are under-studied in APR is mirrored in software engineering more generally [12]; APR is certainly not the exception. Many studies have maligned the industry-academia gap, where tools developed in academia are not widely adopted in industry. The emergence of APR as an important field of research within software engineering represents a key opportunity to ensure that future APR research and development is developer-centred and that the tools developed in coming years fully realise their potential to benefit developers.

## Acknowledgments

We are very grateful for the helpful feedback we received from our reviewers.

## References

[1] X. D. Le, "Towards efficient and effective automatic program repair," in *Proc. 31st IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2016, pp. 876–879.
[2] L. Gazzola, D. Micucci, and L. Mariani, "Automatic software repair: A survey," *IEEE Trans. Softw. Eng.*, vol. 45, no. 1, pp. 34–67, Jan. 2019.
[3] 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 *Proc. IEEE 34th Int. Conf. Softw. Eng.*, 2012, pp. 3–13.
[4] J. Bader, A. Scott, M. Pradel, and S. Chandra, "Getafix: Learning to fix bugs automatically," *Proc. ACM Prog. Lang.*, vol. 3, no. OOPSLA, pp. 1–27, Oct. 2019
[5] N. Abbas and N. Policek, "'don't be the same, be better': An exploratory study on police mobile technology resistance," *Police Pract. Res.*, vol. 22, no. 1, pp. 849–868, 2021.
[6] M. Alohali, F. Carton, and Y. O'Connor, "Investigating the antecedents of perceived threats and user resistance to health information technology: A case study of a public hospital," *J. Decis. Syst.*, vol. 29, no. 1, pp. 27–52, 2020.
[7] W. Weimar, "Program repair, patch quality, and human factors," keynote at 2nd Int. Workshop Automat. Program Repair, May 2021.
[8] B. A. Kitchenham *et al.*, "Preliminary guidelines for empirical research in software engineering," *Main*, vol. 28, no. 8, pp. 721–734, 2002.
[9] M. Monperrus, "The living review on automated program repair," HAL/archives-ouvertes.fr, Lyon, France, Tech. Rep. hal-01956501, 2018.
[10] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting empirical methods for software engineering research," in *Guide to Advanced Empirical Software Engineering*, (Guide to advanced empirical software engineering Series). Berlin, Germany: Springer, 2008, pp. 285–311.

[11] H. Robinson, J. Segal, and H. Sharp, "Ethnographically-informed empirical studies of software practice," *Informat. Softw. Technol.*, vol. 49, no. 6, pp. 540–551, 2007.

[12] M. Storey, N. A. Ernst, C. Williams, and E. Kalliamvakou, "The who, what, how of software engineering research: A socio-technical framework," *Empirical Softw. Eng.*, vol. 25, pp. 4097–4129, 2020.

[13] M. Perscheid, B. Siegmund, M. Taeumel, and R. Hirschfeld, "Studying the advancement in debugging practice of professional software developers," *Softw. Qual. J.*, vol. 25, no. 1, pp. 83–110, 2017.

[14] R. P. Buse, C. Sadowski, and W. Weimer, "Benefits and barriers of user evaluation in software engineering research," *SIGPLAN Notices*, vol. 46, no. 10, pp. 643–656, Oct. 2011.

[15] M. Monperrus, "Automatic software repair: A bibliography," *ACM Comput. Surv.*, vol. 51, no. 1, Jan. 2018.

[16] K. Liu *et al.*, "A critical review on the evaluation of automated program repair systems," *J. Syst. Softw.*, vol. 171, 2021, Art. no. 110817.

[17] S. Beecham, N. Baddoo, T. Hall, H. Robinson, and H. Sharp, "Motivation in software engineering: A systematic literature review," *Informat. Softw. Technol.*, vol. 50, no. 9, pp. 860–878, 2008.

[18] A. Soomro, N. Salleh, E. Mendes, J. Grundy, G. Burch, and A. Nordin, "The effect of software engineers' personality traits on team climate and performance: A systematic literature review," *Informat. Softw. Technol.*, vol. 73, pp. 52–65, 2016.

[19] P. Lenberg, R. Feldt, and L. G. Wallgren, "Behavioral software engineering: A definition and systematic literature review," *J. Syst. Softw.*, vol. 107, pp. 15–37, 2015.

[20] D. S. Cruzes and T. Dyba, "Research synthesis in software engineering: A tertiary study," *Informat. Softw. Technol.*, vol. 53, no. 5, pp. 440–455, 2011.

[21] 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," 2021, *arXiv:2108.13064*.

[22] G. M. Alarcon *et al.*, "Would you fix this code for me? Effects of repair source and commenting on trust in code repair," *Systems*, vol. 8, no. 1, pp. 1–17, 2020. [Online]. Available: https://www.mdpi.com/2079–8954/8/1/8

[23] T. Durieux, F. Madeiral, M. Martinez, and R. Abreu, "Empirical review of java program repair tools: A large-scale experiment on 2,141 bugs and 23,551 repair attempts," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2019, pp. 302–313.

[24] S. Wang *et al.*, "Automated patch correctness assessment: How far are we?," in *Proc. 35th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2020, pp. 968–980.

[25] K. Liu *et al.*, "A critical review on the evaluation of automated program repair systems," *J. Syst. Softw.*, vol. 171, 2021, Art. no. 110817.

[26] X. Huang, H. Zhang, X. Zhou, M. A. Babar, and S. Yang, "Synthesizing qualitative research in software engineering: A critical review," in *Proc. 40th Int. Conf. Softw. Eng.*, 2018, pp. 1207–1218.

[27] K.-J. Stol, P. Ralph, and B. Fitzgerald, "Grounded theory in software engineering research: A critical review and guidelines," in *Proc. 38th Int. Conf. Softw. Eng.*, 2016, pp. 120–131.

[28] H. Zhang, X. Huang, X. Zhou, H. Huang, and M. A. Babar, "Ethnographic research in software engineering: A critical review and checklist," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2019, pp. 659–670.

[29] D. Bowes, T. Hall, and S. Beecham, "SLuRp: A tool to help large complex systematic literature reviews deliver valid and rigorous results," in *Proc. 2nd Int. Workshop Evidential Assessment Softw. Technol.*, 2012, pp. 33–36.

[30] D. S. Cruzes and T. Dyba, "Recommended steps for thematic synthesis in software engineering," in *Proc. Int. Symp. Empirical Softw. Eng. Meas.*, 2011, pp. 275–284.

[31] B. Kitchenham *et al.*, "Trends in the quality of human-centric software engineering experiments–A quasi-experiment," *IEEE Trans. Softw. Eng.*, vol. 39, no. 7, pp. 1002–1017, Jul. 2013.

[32] W. Weimer, "Patches as better bug reports," in *Proc. 5th Int. Conf. Generative Prog. Compon. Eng.*, 2006, pp. 181–190.

[33] M. Monperrus, S. Urli, T. Durieux, M. Martinez, B. Baudry, and L. Seinturier, "Human-competitive patches in automatic program repair with repairnator," pp. 1–3, 2018, *arXiv:1810.05806*.

[34] D. Marcilio, C. A. Furia, R. Bonifácio, and G. Pinto, "Automatically generating fix suggestions in response to static code analysis warnings," in *Proc. 19th Int. Work. Conf. Source Code Anal. Manipulation*, 2019, pp. 34–44.

[35] T. Dybå and T. Dingsøyr, "Empirical studies of agile software development: A systematic review," *Informat. Softw. Technol.*, vol. 50, no. 9, pp. 833–859, 2008.

[36] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *Biometrics*, vol. 33, no. 1, pp. 159–174, 1977. [Online]. Available: http://www.jstor.org/stable/2529310

[37] B. Kitchenham *et al.*, "Can we evaluate the quality of software engineering experiments?," in *Proc. ACM-IEEE Int. Symp. Empirical Softw. Eng. Meas.*, 2010, pp. 1–8.

[38] C. Kern and J. Esparza, "Automatic error correction of java programs," in *Formal Methods for Industrial Critical Systems*, S. Kowalewski and M. Roveri, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 67–81.

[39] X. B. D. Le, "Overfitting in Automated Program Repair: Challenges and Solutions," Ph.D. dissertation, Singapore Manage. Univ., 2018.

[40] V. Balachandran, "Fix-it: An extensible code auto-fix component in review bot," in *Proc. IEEE 13th Int. Work. Conf. Source Code Anal. Manipulation*, 2013, pp. 167–172.

[41] J. P. Cambronero, J. Shen, J. Cito, E. Glassman, and M. Rinard, "Characterizing developer use of automatically generated patches," 2019, *arXiv:1907.06535*.

[42] B. Daniel, V. Jagannath, D. Dig, and D. Marinov, "ReAssert: Suggesting repairs for broken unit tests," in *Proc. 24th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2009, pp. 433–444.

[43] Z. P. Fry, B. Landau, and W. Weimer, "A human study of patch maintainability," in *Proc. Int. Symp. Softw. Testing Anal.*, 2012, pp. 177–187.

[44] S. Gulwani, I. Radiček, and F. Zuleger, "Automated clustering and program repair for introductory programming assignments," in *Proc. 39th ACM SIGPLAN Conf. Prog. Lang. Des. Implementation*, 2018, pp. 465–480.

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

[46] S. Kaleeswaran, V. Tulsian, A. Kanade, and A. Orso, "Minthint: Automated synthesis of repair hints," in *Proc. Int. Conf. Softw. Eng.*, 2014, pp. 266–276.

[47] A. Kalyanpur, B. Parsia, E. Sirin, and B. Cuenca-Grau , "Repairing unsatisfiable concepts in OWL ontologies," *Semantic Web, Res. Appl.*, vol. 4011, pp. 170–184, 2006.

[48] D. Kim, J. Nam, J. Song, and S. Kim, "Automatic patch generation learned from human-written patches," in *Proc. Int. Conf. Softw. Eng.*, 2013, pp. 802–811.

[49] J. Lee, D. Song, S. So, and H. Oh, "Automatic diagnosis and correction of logical errors for functional programming assignments," *Proc. ACM Program. Lang.*, 2018, pp. 1–30.

[50] C. Liu, J. Yang, L. Tan, and M. Hafiz, "R2Fix: Automatically generating bug fixes from bug reports," in *Proc. Int. Conf. Softw. Testing Verification Validation*, 2013, pp. 282–291.

[51] S. Mahajan, N. Abolhassani, P. McMinn, and W. G. Halfond, "Automated repair of mobile friendly problems in web pages," in *Proc. 40th Int. Conf. Softw. Eng.*, 2018, pp. 140–150.

[52] S. Mahajan, A. Alameer, P. McMinn, and W. G. Halfond, "Automated repair of internationalization presentation failures in web pages using style similarity clustering and search-based techniques," in *Proc. IEEE Int. Conf. Softw. Testing Verification Validation*, 2018, pp. 215–226.

[53] Y. Tao, J. Kim, S. Kim, and C. Xu, "Automatically generated patches as debugging aids: A human study," in *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2014, pp. 64–74.

[54] Y. Tomida, Y. Higo, S. Matsumoto, and S. Kusumoto, "Visualizing code genealogy: How code is evolutionarily fixed in program repair?," in *Proc. Work. Conf. Softw. Visual.*, 2019, pp. 23–27.

[55] J. Yi, U. Ahmed, A. Karkare, S. Tan, and A. Roychoudhury, "A feasibility study of using automated program repair for introductory programming assignments," in *Proc. 11th Joint Meeting Found. Softw. Eng.*, 2017, pp. 740–751.

[56] M. Beller, N. Spruit, D. Spinellis, and A. Zaidman, "On the dichotomy of debugging behavior among programmers," in *Proc. 40th Int. Conf. on Softw. Eng.*, 2018, pp. 572–583.

[57] 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 *Proc. 11th Joint Meeting Found. Softw. Eng.*, 2017, pp. 117–128.

[58] M. Monperrus, "Explainable software bot contributions: Case study of automated bug fixes," in *Proc. 1st Int. Workshop Bots Softw. Eng.*, 2019, pp. 12–15.

[59] R. van Tonder and C. Le Goues , "Towards s/engineer/bot: Principles for program repair bots," in *Proc. IEEE/ACM 1st Int. Workshop Bots Softw. Eng.*, 2019, pp. 43–47.

[60] C. Le Goues, T. Nguyen, S. Forrest, and W. Weimer, "GenProg: A generic method for automatic software repair," *IEEE Trans. Softw. Eng.*, vol. 38, no. 1, pp. 54–72, Jan./Feb. 2012.

[61] J. Singer, T. Lethbridge, N. Vinson, and N. Anquetil, "An examination of software engineering work practices," in *Proc. Conf. Centre Adv. Stud. Collaborative Res.*, 1997, Art. no. 21.

[62] D. Falessi *et al.*, "Empirical software engineering experts on the use of students and professionals in experiments," *Empirical Softw. Eng.*, vol. 23, no. 1, pp. 452–489, Feb. 2018.

[63] R. Feldt *et al.*, "Four commentaries on the use of students and professionals in empirical software engineering experiments," *Empirical Softw. Eng.*, vol. 23, pp. 3801–3820, 2018.

[64] P. Ralph *et al.*, "ACM SIGSOFT empirical standards," 2020, *arXiv:2010.03525*.

[65] S. Kirbas *et al.*, "On the introduction of automatic program repair in bloomberg," *IEEE Softw.*, vol. 38, no. 4, pp. 43–51, Jul./Aug. 2021.

[66] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Predicting the location and number of faults in large software systems," *IEEE Trans. Softw. Eng.*, vol. 31, no. 4, pp. 340–355, Apr. 2005.

[67] H. Sharp, Y. Dittrich, and C. B. de Souza, "The role of ethnographic studies in empirical software engineering," *IEEE Trans. Softw. Eng.*, vol. 42, no. 08, pp. 786–804, Aug. 2016.

[68] H. Lieberman, "The debugging scandal and what to do about it (introduction to the special section)," *Commun. ACM*, vol. 40, no. 4, pp. 26–29, 1997.

[69] M. Fowler, *Refactoring: Improving the Design of Existing Code.* Boston, MA, USA: Addison-Wesley, 1999.

[70] V. Ivanov, A. Rogers, G. Succi, J. Yi, and V. Zorin, "What do software engineers care about? Gaps between research and practice," in *Proc. 11th Joint Meeting Found. Softw. Eng.*, 2017, pp. 890–895.

[71] L. Briand, "Embracing the engineering side of software engineering," *IEEE Softw.*, vol. 29, no. 4, pp. 96–96, Jul./Aug. 2012.

[72] M. Endres, G. Sakkas, B. Cosman, R. Jhala, and W. Weimer, "Infix: Automatically repairing novice program inputs," in *Proc. 34th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2019, p. 399–410.

[73] M. Petre, "Uml in practice," in *Proc. Int. Conf. Softw. Eng.*, 2013, pp. 722–731.

[74] D. van der Linden *et al.*, "Schrödinger's security: Opening the box on app developers' security rationale," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, 2020, pp. 149–160.

**Emily Winter** is currently a senior research associate with Lancaster University, specialising in the socio-technical dimension of software engineering. A sociologist by background (PhD, Lancaster University, 2017), her research interests include the perceptions and attitudes of software developers about the technologies that they build and the tools that they use.

**Vesna Nowack** received the PhD degree in computer architecture from Universitat Politècnica de Catalunya, Spain, in 2016. In 2017, she became a teaching assistant with Technische Universität Dresden, Germany. Since June 2019, she has been a postdoctoral researcher with the Queen Mary University of London, U.K. and Lancaster University, U.K. Her current research focuses on automatic program repair, in particular genetic improvement, generation of fix patterns and application of repair tools in industry.

**David Bowes** is currently a senior lecturer of computer science with Lancaster University. He 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. He has a deep knowledge of analysis methods, having built many defect prediction models.

**Steve Counsell** received the PhD degree from the University of London, in 2002. He is currently a professor of software engineering with the Department of Computer Science, Brunel and the head of Brunel Software Engineering Laboratory (BSEL). He has authored or coauthored more than 190 research papers on topics, including data mining, software refactoring, software evolution, and defect analysis. He is a fellow of British Computer Society and was a software developer in industry prior to academia. He has worked extensively on large research projects with industry in the past.

**Tracy Hall** is currently a professor with Lancaster University. Her research interests include software engineering, code analysis, and defect prediction.

**Sæmundur Haraldsson** is currently a lecturer with the University of Stirling. He has co-organised every tutorial on Genetic Improvement with GECCO, PPSN, and CEC. He has coauthored 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.

**John Woodward** received the BSc degree in theoretical physics, the MSc degree in cognitive science, and the PhD degree in computer science from the University of Birmingham, U.K. He is currently with School of Electronic Engineering and Computer Science, Queen Mary University of London U.K., where he is the head of 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.