Towards Developer-Centered Automatic Program Repair: Findings from Bloomberg

Emily Rowan Winter
Lancaster University
Lancaster, UK

Vesna Nowack
Lancaster University
Lancaster, UK

David Bowes
Lancaster University
Lancaster, UK

Steve Counsell
Brunel University London
London, UK

Tracy Hall
Lancaster University
Lancaster, UK

Sæmundur Haraldsson
University of Stirling
Stirling, UK

John Woodward
Queen Mary University of London
London, UK

Serkan Kirbas
Bloomberg
London, UK

Etienne Windels
Bloomberg
London, UK

Olayori McBello
Bloomberg
London, UK

Abdurahman Atakishiyev
Bloomberg
London, UK

Kevin Kells
Bloomberg
New York, USA

Matthew Pagano
Bloomberg
Princeton, USA

ABSTRACT

This paper reports on qualitative research into automatic program repair (APR) at Bloomberg. Six focus groups were conducted with a total of seventeen participants (including both developers of the APR tool and developers using the tool) to consider: the development at Bloomberg of a prototype APR tool (Fixie); developers' early experiences using the tool; and developers' perspectives on how they would like to interact with the tool in future. APR is developing rapidly and it is important to understand in greater detail developers' experiences using this emerging technology. In this paper, we provide in-depth, qualitative data from an industrial setting. We found that the development of APR at Bloomberg had become increasingly user-centered, emphasising *how* fixes were presented to developers, as well as particular features, such as customisability. From the focus groups with developers who had used Fixie, we found particular concern with the pragmatic aspects of APR, such as how and when fixes were presented to them. Based on our findings, we make a series of recommendations to inform future APR development, highlighting how APR tools should 'start small', be customisable, and fit with developers' workflows. We also suggest that APR tools should capitalise on the promise of repair bots and draw on advances in explainable AI.

ESEC/FSE '22, November 14–18, 2022, Singapore, Singapore
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9413-0/22/11.
https://doi.org/10.1145/3540250.3558953

CCS CONCEPTS

- **Software and its engineering**;

KEYWORDS

automatic program repair, human factors, qualitative methods

ACM Reference Format:
Emily Rowan Winter, Vesna Nowack, David Bowes, Steve Counsell, Tracy Hall, Sæmundur Haraldsson, John Woodward, Serkan Kirbas, Etienne Windels, Olayori McBello, Abdurahman Atakishiyev, Kevin Kells, and Matthew Pagano. 2022. Towards Developer-Centered Automatic Program Repair: Findings from Bloomberg. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), November 14–18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3540250.3558953

1 INTRODUCTION

Automatic program repair (APR) is a rapidly-growing area of software engineering involving the automatic generation of patches to fix code defects and bugs. APR has significant potential for reducing the time developers spend manually fixing bugs and freeing up time for other activities. APR research is currently highly technical in its focus, considering potential learning techniques and the types of fixes that can be generated. However, much less is known about developers' experience using APR, and how professional developers want to interact with APR tools. Our prior work found that less than 7% of APR papers feature any research with human participants, and even fewer research with professional developers [28].

Industrial application of APR is currently nascent; Bloomberg (alongside Facebook [2]) is one of few companies where APR has been at least partially implemented. Given the currently limited knowledge of developer experiences using APR, we conducted exploratory focus group research at Bloomberg. This research was conducted as part of an intensive, longstanding collaboration with Bloomberg, in which the first and second authors were seconded to Bloomberg and embedded within a Bloomberg team.

Expanding our knowledge of developers' experiences and perceptions of APR is vital for successful industrial adoption of APR. Within software engineering generally, the gap between academic advances and their industrial take-up has been highlighted [4] [12]. Advances in APR also bring distinct challenges and opportunities. For successful APR exploitation, developers need to accept the automation of their previously manual bug 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, manual bug-fixing) and replaced with other tasks (for example, providing APR tools with specifications [15] or verifying automatically generated patches [6]). In addition, by reducing the time needed for manual bug fixing, APR may free up time for other tasks, leading to restructuring of developer workloads and activities. Overall, APR tools and techniques are likely to change developers' working practices and workflow, potentially having an impact on significant human factors such as developer job satisfaction and motivation. Developing APR tools and techniques that are acceptable to developers is critical to successfully capitalising on the benefits that APR promises, so it is essential that developer experiences using APR are considered.

Our research questions are:

- RQ1: What user considerations have influenced the development of APR at Bloomberg?
- RQ2: What have been the early experiences of developers using Fixie (a prototype APR tool developed at Bloomberg in collaboration with academic researchers)?
- RQ3: How do developers want to interact with Fixie?
- RQ4: What are the lessons learnt from developers' experiences using Fixie for future APR development?

This paper makes the following contributions:

- To the best of our knowledge, we provide the first analysis from in-depth qualitative research of developers' experiences using APR in an industrial setting
- Based on our findings, we present recommendations to inform future APR tool development and hopefully lead to more effective take-up of APR within industry.

The rest of the paper is structured as follows. We provide background information about the development of APR at Bloomberg in Section 2. Our methods and findings are reported in Sections 3 and 4, respectively. Sections 5 and 6 provide recommendations and a discussion of threats to validity. We report on related work in Section 7. Finally, we conclude in Section 8.

2 BACKGROUND: AUTOMATIC PROGRAM REPAIR AT BLOOMBERG

2.1 Developing an APR Tool

Bloomberg is one of few companies where APR has been implemented. A small Bloomberg team has been developing a prototype APR tool called "Fixie" in collaboration with academic researchers. The team was motivated to explore APR by the prevalence of repeated, small bugs/refactorings that Bloomberg developers were having to fix manually, taking up a lot of time with repetitive work. The team hoped to develop an APR system that targeted these small bugs and decrease the manual load on developers. This approach differed from many academic approaches that aim to automatically generate fixes for ever more complex and challenging bugs. Bloomberg's approach was based much more around 'easy wins' that nonetheless are seen to offer significant benefit to developers, removing manual bug-fixing tasks and freeing up developer time. Our previous work [17] further outlines the approach taken by Bloomberg and the industry-academia differences this sustained collaboration brought to light.

In its current form, Fixie provides three different types of fixes: one, off-the-shelf fixes provided by third-party tools (e.g., clang-tidy); two, custom fixes — fixes that are provided by Bloomberg developers for application to other code bases; and three, fixes learnt through version control history. While only the last of these would typically be seen as APR, all three types of fixes enable a more automated bug fixing approach. Custom fixes, for example, allow for automatic application of particular fixes at scale across repos, rather than the same bugs or defects being fixed manually in a more *ad hoc* fashion. Custom fixes have been particularly useful for enabling deprecated components to be updated.

The Fixie architecture (see Figure 1) involves several components. **Fixie-learn** is the learning part of the system able to generate fix patterns automatically from version control history and other data sources. The generation of fix patterns in Fixie-learn occurs through Bloomberg's implementation of the GumTree [10] and anti-unification [19] algorithms (also used by Facebook's Getafix [2]). Fixie-learn takes a code change (as a commit from version control history) and finds the differences between two ASTs (before and after the code change). To generate a fix pattern, Fixie-learn takes a pair of code changes (two commits), anti-unifies them and extracts the richest AST fix pattern that can reproduce both original code changes. A fix pattern is composed of a before pattern and an after pattern and it is chosen as a fix candidate if the before pattern matches a targeted part of code. The situation requiring attention can be any chunk of code, but is usually in the context of an error code or lint warning, as opposed to a program crash; this can significantly improve the outcome, by filtering the fix patterns to only those learned from such situations. Multiple fix candidates for the same targeted code need to be ranked. Unlike Getafix (where ranking is based on the relevance of the fix patterns to the code changes they were generated from), Fixie-learn ranks the fix candidates according to their success in producing a correct fix in the past. The fix generated from the top fix pattern is then provided as a report to software developers who view one fix at a time, each of which they can then either accept or reject. **Fixie-apply** offers fixes to developers in the form of pull requests (PRs). **Fixie-analytics** measures software engineers' acceptance of fixes provided by Fixie-apply.

2.2 Academia-Industry Collaboration

This research came out of a sustained collaboration with Bloomberg. Their interest in APR had been prompted by early collaboration with academic researchers, and this relationship was then consolidated by the first and second authors (researchers at Lancaster University) being seconded to Bloomberg for a year, working closely

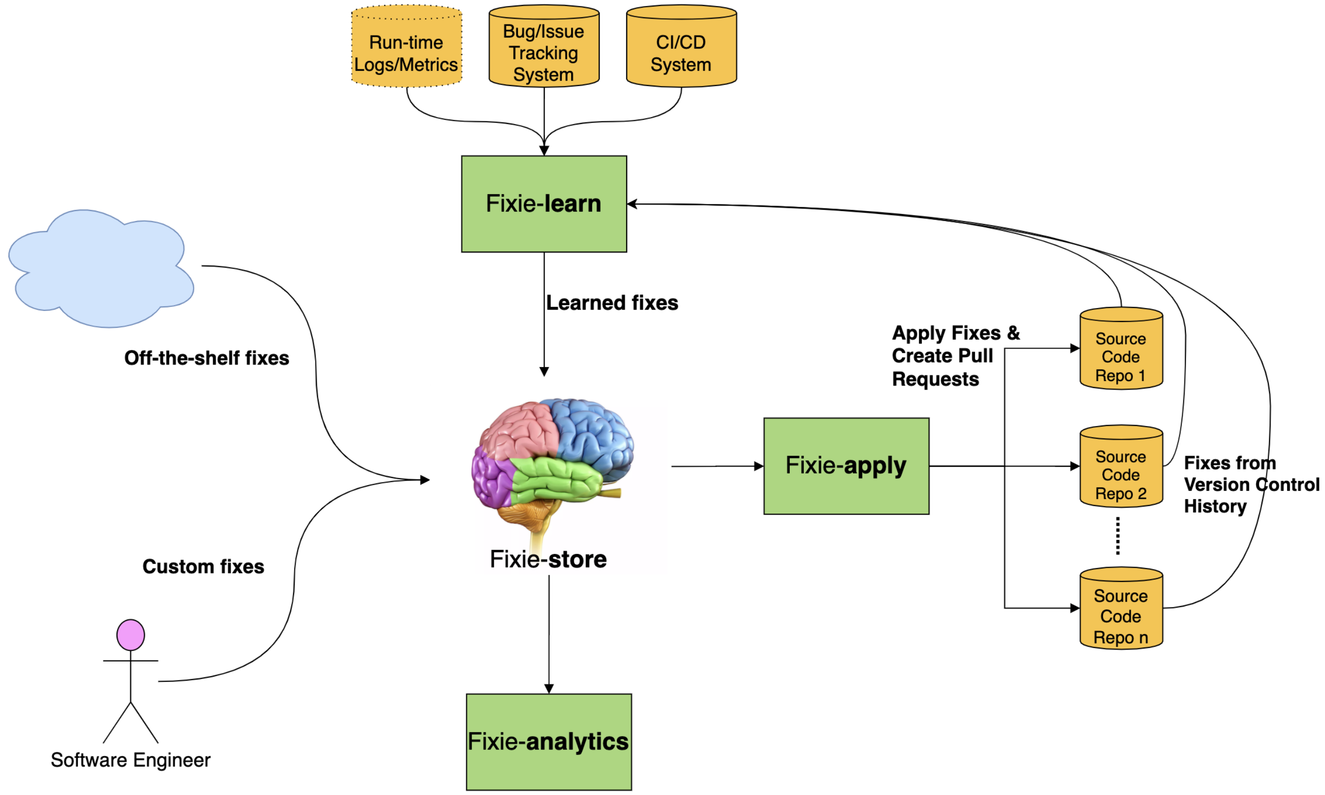


Figure 1: The Fixie architecture

with a small Bloomberg team developing APR technologies. The two researchers became embedded in the team, one focusing on the technical side, and the other on researching the socio-technical dimension. The researchers met with the team regularly, usually two or three times a week, and had access to Bloomberg platforms.

3 METHODOLOGY

3.1 Choice of Focus Group Method

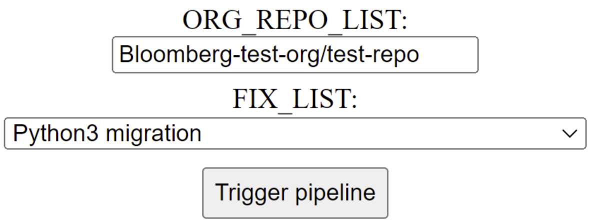
For this research, we chose to use focus groups, a qualitative method well suited to explorative research. Focus groups are not widely used in software engineering, though they are common in marketing and social sciences. One exception is [18], which reports on the use of focus groups for requirements prioritisation and usability evaluation studies. Kontio et al. identify several strengths of focus groups: facilitating discussion of new and unexpected insights; providing the researcher with in-depth understanding; and aiding participant recall and being generally beneficial for participants.

Though Kontio et al. consider the existence of group dynamics a challenge, another key benefit of focus groups is the insight they offer into interaction and negotiation of perspectives. Given the importance of team dynamics and group norms in software development practice [25], we consider focus groups' insight into group dynamics as especially valuable in software engineering research. In particular, focus groups that bring together participants who already know each other (e.g., software teams) may be able to capture group dynamics-related factors that impact tool adoption. An individual's attitudes do not exist in a vacuum and are likely to be shaped by colleagues' views [8]. In our focus groups, there were several instances in which participants changed their minds about a topic or, following discussion, reached agreement. This shows how attitudes are not static, but dynamic and contextual. Understanding these group dynamics may be helpful in contextualising tool adoption.

Focus groups that bring together participants who know each other may be more comfortable and familiar for developers to participate in. Team-based focus groups might also feel similar to software development process activities, such as retrospectives. In both focus groups and retrospectives, for example, individuals share their views on the strengths and weaknesses of a past experience.

3.2 Focus Group Structure and Participants

The focus groups were semi-structured — there were some predetermined questions, but also openness to follow-up on unexpected themes. The focus groups were also adapted to the particular nature of the participants. One focus group was carried out with the Bloomberg team developing Fixie, in order to understand their intentions and goals. The other four focus groups were carried out with different teams who had at least some interaction with Fixie,



ORG_REPO_LIST: Bloomberg-test-org/test-repo

FIX_LIST: Python3 migration

Figure 2: Prototype dashboard for applying fixes across multiple repos

though the extent of this interaction varied. These differences in focus group composition meant that the questions differed between focus groups.

The focus group with the Bloomberg team developing Fixie started with an initial introduction by one researcher and was then focused around the following key open-ended questions:

- What were the origins of Fixie at Bloomberg?
- What were your original aims? Did these aims change?
- How would you like developers to use and interact with Fixie?
- What is your vision for the future?

The other focus groups were tailored to the specific user group; each group had a slightly different experience with Fixie (see Section 4.2). However, each focus group followed a similar structure.

- **Introduction** to the research and the focus group.
- **Section 1:** Questions about participant knowledge of and experience with Fixie, including benefits and challenges experienced so far.
- **Section 2:** Questions about how participants would like to interact with Fixie in its current form (provision of fixes as PRs), including how and when participants would like to be informed about fixes.
- **Section 3:** Questions about how participants would like to interact with Fixie under different possible future directions. Participants were shown a prototype dashboard (see Figure 2), which would allow developers to apply a particular fix at scale across multiple repos. Participants were also prompted about other possible Fixie future developments, such as Fixie as an automated reviewer of developers' own PRs.
- **Section 4:** The focus groups concluded by asking participants to consider the opportunities and challenges for rolling out Fixie across Bloomberg more widely.

The focus groups were facilitated by the first author who has several years' experience in conducting qualitative research, including focus groups. The second author also attended to aid in facilitation and, given her close involvement in Fixie's technical development, to answer more technical questions that participants might have. This integration of social and technical expertise was useful for carrying out the research.

Table 1: Focus group participants

Focus group	Relationship with Fixie	Participants
A	Developers of Fixie	A1, A2, A3
B	Providers of custom fixes	B1, B2, B3
C	Security experts	C1, C2
D	Users of Fixie	D1, D2
E	Users of Fixie	E1, E2, E3, E4, E5
F	Users of Fixie	F1, F2

Potential focus group participants were identified through discussion between the researchers and members of the Bloomberg Fixie team. Participants were then contacted by email or Bloomberg's instant messaging platform. All participants were given an information sheet about the research and returned a consent form. The project had approval from the researchers' University Research Ethics Committee and Bloomberg's Legal and Compliance teams.

Each focus group was made up of people who knew each other, recruited through their teams. Table 1 outlines each of the participants, their team, and the team's relationship with Fixie.

Each focus group took place online, using Bloomberg's video conferencing software, and was audio recorded. Each focus groups lasted between one and one and a half hours, and together they yielded over 40,000 words of transcribed material.

3.3 Focus Group Analysis

The focus groups were transcribed by the first author. They were then thematically coded. The first and second authors read the transcripts and then met to discuss coding. It was decided initially to apply broad brushstroke coding by role and time. The thematic codes for role were: developing (related to developing Fixie); contributing (related to contributing to Fixie, e.g., providing custom fixes); and using (relating to receiving fixes from Fixie). The time codes were past, present, future, and general. Both authors then independently coded three transcripts with these broad codes, while also using open coding to identify emerging and more granular themes. The unit of analysis was generally each separate unit of speech, i.e., what a person said before another person spoke. If speech was more fragmented (i.e., participants said just a few words each), we thematically coded multiple units of speech.

After both authors had independently coded three transcripts, we met to discuss disagreements and establish agreement. The emergent thematic codes were finalised and written up in a code book. We then coded the remaining three transcripts using the code book as reference but continued to use open coding to capture further emerging themes. We then met again to negotiate any disagreement and agree upon new codes. All transcripts were dual-coded independently and there was at least partial agreement in the case of 77.5% units of analysis coded by both authors.

To aid analysis, the transcripts were then uploaded into NVivo, thematic analysis software. This enabled quotations attached to the same themes to be clustered, facilitating further organisation of the data. From these clusters, key findings were identified.

4 FINDINGS

In this section, we report on our findings for RQs 1, 2 and 3. Our recommendations in response to RQ4 are reported in Section 5. For each RQ, our key findings and indicative supporting quotations are also presented in a table (see Tables 2, 3, and 4).

4.1 RQ1: What User Considerations Have Influenced the Development of APR at Bloomberg?

The key finding from focus group A (with the developers of Fixie) was that **APR development had become increasingly user-centered**, representing a shift away from a predominantly technical approach. A3 identified a *'huge mindset shift'*: *'[we] realised that if we want this to go into production and actually be used by developers, this [technical details] shouldn't be the only focus [...] We have to find a way to make developers trust the fixes and to know [...] when we should provide fixes'*.

One key feature of the user-centered approach was an emphasis on putting Fixie users in control: *'if we put the developer in the driving seat, we believe that it will increase interaction and also the confidence of the developer in the solution'* (A1). The team was open to Fixie being shaped by the developers using it: *'developers are shaping [Fixie] in [...] the way they want to interact'*.

A user-centered approach was characterised, in practical terms, by a strong focus on how and when fixes are presented to developers. As A2 commented, *'what's very important is about how and when changes are presented to the developer. Maybe that's even more important than the correctness of the change in some way'*. This is in contrast to much academic research that targets fix correctness as a key objective. Our RQ2 and RQ3 findings confirm this: developers were happy to use Fixie-provided fixes as 'starting points', rather than needing them to be fully accurate, but they needed fixes to be presented to them in the right way.

The provision of relevant information and metrics alongside fixes was seen as important to aid developers reviewing the fix. This was especially significant *'when a fix is suggested but the developers don't really understand what it's supposed to do or how it will interact in the code'* (A3). In such cases, it was important to provide *'very concrete and empirical metrics to show that this is actually a good modification, so the most obvious example would be to fix a test in that case. So if developers know that their current code is failing a test and with that fix the test is now working they will have direct proof that this is actually solving something'* (A3).

Several future directions for Fixie development were identified in focus group A. **One future direction is Fixie as a developer's assistant**, potentially using speech recognition or *'agent-based'* models to enable the tool to respond to *'high-level objectives'* (A1). This vision corresponds with nascent APR research on repair bots [27] [22]. We discuss this further in Section 5.

Another future direction identified in focus group A was to enhance Fixie's customisability, allowing developers to customise Fixie according to their needs. This could be based upon developers' understanding of other compensating controls in their environment that reduce the risk being presented. A3 echoed that Fixie should be customisable according to how open developers are *'to new ideas and new features'*. This suggests that human values are important for how developers want to interact with an APR tool.

One other future possibility identified was to develop **Fixie as a PR reviewer**, motivated by the fact that *'it is a very natural place to interact with people, and to raise the problems and also relationship between a fix and the problem detection'* (A1). A3 agreed: *'PRs are good [...] because you know that the developer has been working on some parts and you know that if you suggest a fix on this part they will be very likely to engage with the suggestion, so I think that's a very good time'*. A3 highlighted how this meant they would be able to provide *'fixes on what was written during the last modification so that it doesn't even get to the code base'*.

Currently, Fixie operates as part of CI processes, fixes being offered as PRs. Focus group A also discussed **implementing Fixie in the IDE** as an alternative future direction. A2 explained, *'so right now [...] you still have to push your changes before Fixie will get involved'*. A3 agreed: *'I think that 95% of bug fixing goes on before reaching the commit phase [. . .] that means we are missing a lot of potential bug fixes'*. From a technical perspective, an IDE-based system could expand Fixie's learning context to incorporate potentially more valuable fixes: *'right now [...] most of the fixes we are trying to address are mainly things that aren't code-breaking [...], so if we [were] seeing the entire story of the developer working locally, and modifying their code to fix tests, we would have much more patterns I think, and much more useful ones as well, because we could actually provide fixes that do fix a test and not just marginally improve the code'* (A3). An IDE-based system could also offer better **user workflow fit**, as A2 explained: *'if I own a code base but I'm not working on it and someone suggests a PR, I'm going to say 'why do we need this change now?' [...] whereas if I'm working on something and someone suggests something, even if it's not as important, even if it's not as correct, I'm more willing to pay more attention to it because I see it as adding more value to what I'm currently doing'*.

4.2 RQ2: What Have Been Developers' Early Experiences Using Fixie?

The experiences developers had with Fixie varied between focus groups, related to the role of participants. Participants in focus group B were involved in providing custom fixes to Fixie, while participants in focus group C offered a security perspective based on their roles as senior security experts. Participants in both these focus groups offered more high-level, strategic insight, whilst the participants in focus groups D, E and F had all received fixes from Fixie in the form of pull requests.

One key strategy employed by Fixie developers is **user-goal alignment**. This was implemented partly by pushing fixes generated by Fixie at scale during large events or project milestones, like migration events. These events involved a degree of **gamification** to motivate developers to review PRs, as teams 'competed' against each other over the course of a one-day hackathon. This seems to be something that worked well. A1 explained that *'half of the PRs (at one migration event) were accepted and merged [...] as opposed to 10% in general'* [61 out of 127 changes were accepted - 48%].

Participants who had received PRs from Fixie as part of migration events were generally positive about the experience. However, they also identified that **manual work was still**

Table 2: Summary of key findings for RQ1 - what user considerations have influenced the development of APR at Bloomberg?

Key finding	Indicative quotation(s)
APR development at Bloomberg has become increasingly user-centered	'We have to find a way to make developers trust the fixes' (A3) 'What's very important is how and when changes are presented to the developer' (A2)
Fixie users need to feel in control	'If we put the developer in the driving seat, we believe that it will increase interaction and also the confidence of the developer in the solution' (A1)
It is important to provide relevant information and metrics with fixes to help users review the fix	'Very concrete and empirical metrics to show that this is actually a good modification, so the most obvious example would be to fix a test in that case' (A3)
Fixie users need to be able to customise Fixie to meet their needs	'This could be based upon developers' understanding of other compensating controls in their environment that reduce the risk being presented'(A1)
Ideas for future development include Fixie as a 'developer's assistant' and an IDE-based system	'I imagine maybe Fixie as a developer's assistant' (A1) 'If I'm working on something and someone suggests something [...], I'm more willing to pay more attention to it because I see it as adding more value to what I'm currently doing' (A2)

Table 3: Summary of key findings for RQ2 - What have been developers' early experiences using Fixie?

Key finding	Indicative quotation(s)
Manual work is still required when using Fixie	'I had to manually modify the file generated by Fixie' (F2) 'You still need manual work [...] sometimes you need to fix the unit test and I don't think Fixie can do that for you' (F1)
'Fixie-generated fixes offer a helpful starting point	Fixie has done the heavy lifting [...] even if the PR is not functioning, one can still make some small changes, small fixes, to make it production worthy or improve the style of it' (E5)

required. F1 explained, *'the overall structure of the migration looks good but you still need manual work [...] you need to fix the path [...] and sometimes you need to fix the unit test and I don't think Fixie can do that for you'*. F2 discussed the need to manually modify the code changes generated by Fixie: *'I was using a special version of a library [that] was not detected when Fixie was run, so I had to manually modify the file generated by Fixie, so to point at the correct library'*.

However, **the Fixie-provided fixes still offered a helpful starting point**. E5 explained that Fixie had *'done the heavy lifting [. . .] even if the PR is not functioning, one can still make some small changes, small fixes, to make it production worthy or improve the style of it. So even if the Fixie PRs can do most of the job, so like 90% of the job, but 10% require [a] tweak, that will still be good progress, a good contribution by an automatic tool for me'*. Again, this reveals a **pragmatic** approach to automatically generated fixes, rather than a focus purely on technical precision.

4.3 RQ3: How Do Developers Want to Interact with Fixie?

4.3.1 Fixie-Generated PRs. As Fixie-generated fixes are currently presented to developers as PRs, there was much discussion in focus groups D, E and F of how the PR **workflow** was best managed. Participants identified that within Bloomberg generally (not just related to APR), **there could be a delay getting PRs approved**, particularly for very large changes involving many lines of code.

One implication of this challenging context was that **Fixie PRs should be carefully timed**. B3 explained that *'the best way to guarantee [PR review] is if the team is on board'*, because unexpected PRs might be deprioritised: *'maybe they have much more important*

Table 4: Summary of key findings for RQ3 - How do developers want to interact with Fixie?

Key finding	Indicative quotation(s)
Fixie-generated PRs should be well timed and incorporated into existing processes	'We have a weekly build already, [...] so if at the same time they can run Fixie and then said "oh by the way, Fixie found those things that we would like to change"' (F2)
Processes are needed to support developers if they do not understand a Fixie-generated PR	'I think it did happen a few cases where I saw a Fixie PR and I maybe didn't really understand why it's needed [...] and then I just ignore it, because I don't have anyone to ask' (E1) It would be useful to have 'a link to some Wiki page or whatever of why you are doing the changes' (F2)
Who PRs are assigned to is a key question, provoking different opinions	'The one who merged the last PR' (E3) 'The one who will most want to take action on it [but] how do you find out who is most interested in it?' (E2) The team lead should 'distribute the workload' (E5)
PRs should be small and easy to code review	'If it is relatively small size then I can read it within 5 minutes, then that's a good one' (F1) PRs should be 'easy to code review, so they're simple and you can easily determine that they're safe' (E2)
Automatically merging PRs was a controversial idea for developers	'I wouldn't go for auto-merge in any case' (E5)
Pre-requisites for automatically merging PRs included high test coverage and high confidence	'I think if we have really good test coverage [...] then I would feel more confident in auto-merging' (E2) There would need to be 'a really high confidence rate-like, for the past 6 to 8 months, 99% of fixes were approved without hesitation' (C1).
Fixie should start with small fixes	'The way I would do it is I would start with small things and then you'll build confidence and people will be more and more willing to use it' (F2)
Developer awareness of Fixie activities is paramount	'If there were communication about it and I was aware of the change, then there wouldn't be an issue at all' (E4)

things to do [...] so they will completely deprioritise the Fixie PRs or maybe even close them'. One possible solution was to **incorporate Fixie-generated PRs into existing processes**, as F2 identified: *'we have a weekly build already, [...] so if at the same time they can run Fixie and then said "oh by the way, Fixie found these things that we would like to change; would you be ok?"'*.

The PR-based APR system also raised questions of **what to do if the PR was difficult to understand**. E1 explained, *'if there's an open PR by a developer I either just merge it or, if I'm not sure, I reach out to that person. But in Fixie – I think it did happen a few cases where I saw a Fixie PR and I maybe didn't really understand why it's needed, or why it should be added, and then I just ignore it, because I don't have anyone to ask'*. One solution would be a description of *'what is this for, what is it trying to fix'* (E1), or a *'a link to some Wiki page or whatever of why you are doing the changes'* (F2).

Another key discussion theme was **who PRs should be assigned to**. Participants identified various solutions, including *'the last person who committed to that PR'* (E3) or the *'team lead'* (E2, E3). E2 felt that PRs should be assigned to *'who will most want to take action on it [but] how do you find out who is most interested in it?'*. E2 suggested that assigning PRs to the team lead, who would then assign to a relevant team member, might work: *'you want to involve the team leader somehow so that they're involved in the prioritisation of work'*. E5 agreed that the team lead should *'distribute the workload'*. In another focus group, B3 suggested that PRs should be auto-assigned based on ownership or previous PR reviewers.

As well as who PRs should be assigned to, there was much discussion of **how and when people should be notified about Fixie-generated PRs**. Participants didn't want to be notified about PRs too frequently unless a fix was urgent: *'I'm a bit worried of getting a message every day saying "ok, you have a new PR from Fixie", because there's already a lot of PRs to review'* (F2). E2 agreed that they would want *'one message approximately per week, with a link to the webpage where I can see my matrix of what's going on and I can do my assignments'*. In addition, it was important that participants did not receive an overwhelming number of PRs: E2 explained that developers should *'get only a certain number at a time, that they don't pile up'*.

Other participants were less worried about frequency and number of PRs, and more concerned about the **size of the PRs**, as F1 expressed, *'the frequency doesn't matter that much to me but the size of the PR matters – [...] if it is relatively small size then I can read it within 5 minutes, then that's a good one'*. E3 agreed that PRs should not be too large: *'the only reluctance a developer may run into is if it generates a PR which has changes to let's say hundreds of files'*.

As well as small in size, **PRs should be easy to code review** : *'so they're simple and you can easily determine that they're safe'* (E2). E2 expanded on what would make a PR easy to code review: *'that it does one thing only at a time, that what it's doing is clearly a fix [. . .] and that you know won't have side effects, and that [won't] get in the way of business logic [...] So if Fixie wants to fix an if statement for me, I'll say "ok, I'll postpone that and look at that later" *laughs*, but if Fixie says "actually, the way you've done this, here's the better way to do it, and I look at it and I know that"'oh yeah, this is a transformation that I understand will have no side effects", then I can approve that more quickly'*.

A contentious issue in the focus groups was **whether PRs should ever be automatically merged to a code base**. Some participants felt that it would always or mostly be necessary for a developer to check the fix: *'I wouldn't go for auto-merge in any case'* (E5); *'it does depend very heavily on the change but most of the time it's good to just get one last sanity check before it gets merged'* (E4). Other participants felt that certain circumstances might allow for auto-merge, such as *'very minor changes'* (E1), in circumstances where a change had remained unmerged *'for a long duration'* (E3) or in conditions of very high test coverage: *'I think if we have really good test coverage [...] then I would feel more confident in auto-merging happening'* (E2). Auto-merge had several prerequisites. These included **good communication**: *'I think if there were communication about it and I was aware of the change, then there wouldn't be an issue at all [...] rather than just auto-merge and figure out what happened after'* (E4); and **proven success**: developers need to manually review fixes until there is '*a really high confidence rate-like for the past 6 to 8 months, 99% of fixes were approved without hesitation'* (C1).

Aside from the particular case of auto-merging PRs, high test coverage was seen as generally important for PR acceptance. F1 recommended that PRs were generated first for *'higher coverage repos'*, F2 confirming that *'I have that peace of mind if I have high coverage in regression and unit tests'*.

4.3.2 Trust in Fixie.

The notion of automatically-generated fixes raised various issues related to trust, particularly in the security focus group, whose members were **worried about automatically generated fixes making code less secure**. C2 explained *'my biggest fear [...] if someone fixes something the wrong way and either breaks things or makes it less secure'*, and C1 agreed: *'yes, I was about to say that too [...] not only not fix the vulnerability, but make it less secure in the process'*.

Trust was also raised as an issue in the other focus groups and **starting with small fixes** was seen as an important way to build trust among Fixie users. F2 explained: *'the way I would do it is I would start with small things [...] and then you'll build confidence and people will be more and more willing to use it'*. Another way of building trust was for **developers to have awareness of what Fixie was doing**. E2 talked about the need for developers to have *'situational awareness'* and to have Fixie communicate its actions to the team: *'so it's not so much that we're aware of the syntax changes that happened, but we're aware of the theme, in the same way that we would be aware of something that says 'ok, this week we're building in this or that into the parser'*.

4.3.3 Future Directions - Dashboard.

Participants in focus groups D, E and F were presented with a prototype Fixie dashboard to elicit their responses. The dashboard is a simple web page, which would allow a developer to pick a specific fix type and then apply this at scale to a repo, or to multiple repos. The focus groups provided feedback on the idea itself and its implementation. Regarding the idea, participants were not fully convinced of its value. This was partly because they were unsure about the value of triggering Fixie themselves. Asked if they would be interested in applying a fix for a deprecated API across multiple repos, one participant replied *'but shouldn't Fixie proactively look for old usages of this library and open PRs for other repos in the organisation that have this'* (E1). E1 continued that they didn't see how this would be useful for *'machine generated changes'*: *'if it's machine generated, the machine part should also be proactive in searching where this change is needed and suggesting it'*. The proposed dashboard was only seen as useful for human-created, or 'custom' fixes, not for automatically-generated fixes.

There was also some feedback on the dashboard's interface. E2 suggested that more **visual cues** were needed: *'give me a visual so I understand how important this change is, and how big of a change it is [. . .] I think that will be really helpful to want to interact with it'*. E2 also suggested that a 'try' button was needed as well as an 'apply' button, which would *'let you visually inspect the results and to do it as many times as you want starting from scratch'*. Though the apply button would only generate PRs rather than actually apply the change to the codebase, E2 still felt that a try button would be useful: *'I don't want to generate PRs that aren't what I wanted, aren't what I thought they were, aren't what I expected'*.

4.3.4 Future Directions - Fixie as a PR Reviewer.

As well as the dashboard, we also asked developers what they felt about the idea of **Fixie as a PR reviewer**, receiving a fairly positive response. E4 could see value in this idea if suggestions were related to **code formatting**: *'something that would maybe catch [formatting errors] and give a suggestion for that could be useful'*. E2 was also positive about *'Fixie coming in and saying "ok, I see what you're doing here, there's a better way to do it, or there's a better syntax for that, or here's some options for you"'*, and felt that developers could be incentivised by knowing that Fixie had helped them develop *'great code'* that code reviewers would pick up on. However, not all participants felt that Fixie as a PR reviewer would add value. C1 explained that *'you get already all the benefits by having the [Fixie-generated] PR making the change and then that can be reviewed'*. F2 felt that recommendations about *'trivial things'* could also *'upset'* people.

5 DISCUSSION AND RECOMMENDATIONS

5.1 Developer-APR Interaction

There are several important considerations for developer-APR interaction. One of these is how APR can respond to the problem of what developers should do if they don't understand an automatically generated fix. Advances in explainable AI offer one potential solution, as does the idea of repair bots, which has already had some attention within APR [22] [27]. 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 those questions' [22]. Such solutions also respond to another significant consideration that emerged in the focus groups – how APR communicates its activities and intentions to developers, leading to developer situational awareness.

The focus groups with Fixie users found that the idea of automatically applied fixes was controversial for developers and unpopular with many of them. In addition, the developers of Fixie were clear that it was important for developers to be involved in the process and to feel in control. This suggests that, for the time being at least, APR development should focus upon APR tools that continue to involve the developer in reviewing and approving patches, rather than aiming to remove the developer completely, a goal for some APR research (for example, [29]).

Recommendations:

- APR tools should incorporate explainable AI techniques.
- Repair bots offer an important future step for ensuring effective developer-APR interaction.
- Development of APR should prioritise systems that still involve the developer in reviewing and approving patches.

5.2 Fit with Workflow

The lengthy discussions about how and when PRs should be communicated and to whom they should be assigned reveals that the introduction of APR is not going to be simple. Much work is needed to identify how APR tools can be developed so that developers can incorporate them into their existing workflows. This has similarity with Erlenhov et al.'s work on software bots, which argues that 'such systems need to carefully evaluate how often, and when, they should interrupt the developer with suggestions or requests for further input' [9]. Within the focus groups, there was not always agreement about the best ways in which APR could be incorporated into developer workflows, indicating that there is no simple 'one-size-fits all' model for effective APR. Not only does workflow-fit vary among developers, but 'anything for Bloomberg has to be very tailored to Bloomberg', complementing its existing processes (C2).

Recommendation:

- APR tools should be developed so that they are customisable to company workflow processes and developer workflow preferences. This might include developing APR tools that can be used in both CI and IDE contexts

5.3 Developer Trust in APR

Trust in APR was a less frequently occurring theme than discussion of the practicalities of how and when automatically generated fixes should be presented. However, it was still important. Developers expressed the view that an APR tool should 'prove itself' by starting small. Though developers varied in their attitudes towards whether fixes should be ready to apply or a 'starting point' for the developer to modify where needed, there seemed to be general consensus that showing proven success with small, straightforward fixes was desirable for building trust. Whilst fixing complex bugs is a key imperative for academic APR research, offering fixes for small, simple bugs is an important way to build developer trust.

In addition, the different kinds of developer risk appetite identified in focus group A is an important consideration (given other compensating controls), suggesting that some developers prefer fully understandable patches, while others — with greater risk appetite — may respond positively to what Monperrus refers to as 'alien code' [21].

Recommendations:

- APR tool development should consider how fixes could be offered in a 'phased' way, starting with small fixes and becoming more complex. APR tools could learn from acceptance metrics when to start offering more complex fixes, or interact with the developer to ask whether more complex fixes should be offered (see, for example, advances in APR bots [22] [27]).
- APR tools should be customisable according to individual developers' risk appetite.

5.4 Technical Considerations

Whilst the focus of this work was on the socio-technical dimensions, the focus groups brought to light one particularly significant technical dimension. Bloomberg, like many companies, has specific types of bugs, such as bugs that concern business logic. This raises challenges for APR's learning context, as it makes it essential to access proprietary software in order to develop APR tools that are more effective and useful for different industry settings.

Recommendation:

- Sustained academia-industry collaborations are required to develop bespoke APR tools for specific industry settings.

6 THREATS TO VALIDITY

Internal validity: One key threat to validity in focus group research is that it may be challenging to capture participants' 'actual' views outside of the social dynamics and pressures present in a group setting. Whilst we consider the capturing of group dynamics a strength of focus groups — as these group dynamics are likely to influence tool adoption and acceptance — the group setting does pose challenges, such as more dominant participants. To mitigate this threat and ensure that less dominant voices were heard, the focus group facilitator intentionally addressed some people directly for input at various stages during the focus groups.

Another threat to internal validity is that research that investigates people's experiences and attitudes is susceptible to social desirability bias. Our participants may have felt influenced by the presence of their colleagues and of researchers. Knowing that the two researchers were working closely with the development team may have also hindered users in being more critical of Fixie. To mitigate this threat, we tried to make the focus groups informal and relaxed; we also stressed that the data would be anonymous and that only the researchers would have access to the raw transcripts.

To mitigate against interpretive bias in the thematic analysis, each transcript was independently thematically coded by two authors, and disagreements were discussed until agreement was reached.

External validity: Our sample size of 17 is not uncommon for qualitative research [18], and was purposefully chosen to capture insights from both developers of Fixie and developers using Fixie. However, the insights drawn from these focus groups may not apply to all developers at Bloomberg; in a large and complex organisation, many factors may influence attitudes towards a new tool and we would predict different team dynamics and roles to play a part.

This research was embedded in Bloomberg, a specific industrial setting, and explored a specific prototype APR tool. As a result, our findings can not be generalised to different industrial settings. However, we suggest that they are likely to have relevance elsewhere and we invite further studies to investigate this.

7 RELATED WORK

Our previous work found that only a very small proportion of APR research included any type of study with human participants [28]. Past studies include controlled experiments [1] [3] [5] [7] [13] [14] [24] [26] [30] and surveys [11] [16] [20] [23] [24]. These studies primarily focus on the degree to which participants are aided by having access to APR (for example, [7] [24]) and also the degree to which participants trust APR (for example, [1]).

Past experimental studies have largely focused on how participants performed a task with or without access to an APR tool or its automated patches. These studies paint a mixed picture of the effectiveness of participant interaction with APR, and the degree to which APR assisted them in their tasks. Daniel et al., for example, suggest that their participants may have 'become overly reliant on the tool', as more faults were introduced by participants that had access to the APR tool than those that did not [7]. Parnin and Orso's participants were able to perform an easy task quicker when they had access to the tool, but the tool did not help in the case of more difficult tasks [24]. In Cambronero et al.'s study [5], two participant groups were 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. The results found little difference between the two groups in terms of time taken to perform the assigned tasks and the number of correct patches submitted. Cambronero et al. concluded that automatically generated patches on their own were not enough to increase developer 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' [5]. This finding suggests that work needs to be done regarding how automatically generated patches are presented to developers, and what information is required to aid developers in approving patches.

APR human studies have also drawn attention to issues related to trust. In Tao et al.'s study, for example, participants were worried that generated patches might be unclear or incorrect, and and that they might not work if the test suite is not sufficient [26]. The participants in Liu et al.'s study voiced some concern about the accuracy of the patches generated by the tools [20], while Böhme et al.'s study [3] found that 'practitioners are wary of debugging automation', particularly for functional bugs. Böhme et al.'s participants were unsure how feasible APR was, due to the challenges of code comprehension. Alarcon et al.'s experimental study also considered trust in APR, and found that the source of the repair (human vs. automated) had significant influence on trust perceptions and intentions, participants having higher trust in human repairs than automated repairs [1]. These trust issues have implications for APR adoption, and indeed the participants in Parnin and Orso's study 'were quick to disregard the tool if they felt they could not trust the results or understand how such results were computed' [24]. By contrast, Noller et al.'s survey of 103 participants found high willingness from participants to review automatically generated patches [23]. 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.

These studies provide interesting insights, but are all quantitative, providing little, in-depth, qualitative insight into developer perceptions about APR. In addition, few studies have been conducted with professional developers (exceptions are [3], [23] and [24]), and none situated in specific industry settings. A study of the implementation of Getafix at Facebook suggests that auto-fixes should be integrated into existing development tools and predicted fast enough so as not to slow down engineers' work [2], but these recommendations are not based on a research with developers.

8 CONCLUSION

This paper reports on findings from in-depth, qualitative research at Bloomberg, where a prototype APR tool, Fixie, has been developed and implemented. We find that developers using Fixie are highly concerned with the pragmatic aspects of APR, such as how and when fixes are presented to them. Such aspects have so far been given little attention in APR research. From our findings, we recommend that APR tools should be customisable, start small, and be designed with greater consideration of workflow issues. We also identify research in explainable AI and repair bots as useful future directions. We suggest that following these recommendations will help with the adoption of APR in industry, allowing APR's considerable potential benefits to be more fully realised.

ACKNOWLEDGMENTS

This work was supported by an Engineering and Physical Sciences Research Council grant (EP/S005730/1)/ We are very grateful to the Bloomberg developers who participated in our focus groups and gave of their time and expertise.

REFERENCES

[1] Gene M. Alarcon, Charles Walter, Anthony M. Gibson, Rose F. Gamble, August Capiola, Sarah A. Jessup, and Tyler J. Ryan. 2020. Would You Fix This Code for Me? Effects of Repair Source and Commenting on Trust in Code Repair. *Systems* 8, 8 (2020), 1–17. https://doi.org/10.3390/systems8010008

[2] Johannes Bader, Andrew Scott, Michael Pradel, and Satish Chandra. 2019. Getafix: Learning to Fix Bugs Automatically. *Proc. ACM Program. Lang.* 3, OOPSLA, Article 159 (Oct. 2019), 27 pages. https://doi.org/10.1145/3360585

[3] Marcel Böhme, Ezekiel O. Soremekun, Sudipta Chattopadhyay, Emamurho Ugherughe, and Andreas Zeller. 2017. Where is the Bug and How is It Fixed? An Experiment with Practitioners. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (Paderborn, Germany) *(ESEC/FSE 2017)*. Association for Computing Machinery, New York, NY, USA, 117–128. https://doi.org/10.1145/3106237.3106255

[4] L. Briand. 2012. Embracing the Engineering Side of Software Engineering. *IEEE Software* 29, 4 (2012), 96–96. https://doi.org/10.1109/MS.2012.86

[5] José Pablo Cambronero, Jiasi Shen, Jürgen Cito, Elena Glassman, and Martin Rinard. 2019. Characterizing Developer Use of Automatically Generated Patches. *arXiv preprint arXiv:1907.06535* (2019), 1–9. https://doi.org/10.48550/arXiv.1907.06535

[6] Liushan Chen, Yu Pei, and Carlo A. Furia. 2017. Contract-based program repair without the contracts. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 637–647. https://doi.org/10.1109/ASE.2017.8115674

[7] Brett Daniel, Vilas Jagannath, Danny Dig, and Darko Marinov. 2009. ReAssert: Suggesting Repairs for Broken Unit Tests. In *Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering*. 433–444. https://doi.org/10.1109/ASE.2009.17

[8] Fred D. Davis. 1989. Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly* 13, 3 (1989), 319–340. http://www.jstor.org/stable/249008

[9] Linda Erlenhov, Francisco Gomes de Oliveira Neto, and Philipp Leitner. 2020. An Empirical Study of Bots in Software Development: Characteristics and Challenges from a Practitioner's Perspective. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 445–455. https://doi.org/10.1145/3368089.3409680

[10] Jean-Rémy Falleri, Floréal Morandat, Xavier Blanc, Matias Martinez, and Martin Monperrus. 2014. Fine-grained and accurate source code differencing. In *ACM/IEEE International Conference on Automated Software Engineering, ASE '14, Vasteras, Sweden - September 15 - 19, 2014*. 313–324. https://doi.org/10.1145/2642937.2642982

[11] Hideaki Hata, Emad Shihab, and Graham Neubig. 2018. Learning to Generate Corrective Patches using Neural Machine Translation. *arXiv preprint 1812.07170* (2018), 1–20. https://doi.org/10.48550/arXiv.1812.07170

[12] Vladimir Ivanov, Alan Rogers, Giancarlo Succi, Jooyong Yi, and Vasilii Zorin. 2017. What Do Software Engineers Care about? Gaps between Research and Practice. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (Paderborn, Germany) *(ESEC/FSE 2017)*. Association for Computing Machinery, New York, NY, USA, 890–895. https://doi.org/10.1145/3106237.3117778

[13] Shalini Kaleeswaran, Varun Tulsian, Aditya Kanade, and Alessandro Orso. 2014. Minthint: Automated Synthesis of Repair Hints. In *Proceedings of the International Conference on Software Engineering*. Association for Computing Machinery, New York, NY, USA, 266–276. https://doi.org/10.1145/2568225.2568258

[14] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and Bernardo Cuenca-Grau. 2006. Repairing Unsatisfiable Concepts in OWL Ontologies. In *The Semantic Web: Research and Applications*. Vol. 4011. Springer Berlin Heidelberg, Berlin, Heidelberg, 170–184. https://doi.org/10.1007/11762256_15

[15] Besma Khaireddine, Matias Martinez, and Ali Mili. 2019. Program Repair at Arbitrary Fault Depth. In *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*. 465–472. https://doi.org/10.1109/ICST.2019.00056

[16] Dongsun Kim, Jaechang Nam, Jaewoo Song, and Sunghun Kim. 2013. Automatic Patch Generation Learned from Human-Written Patches. In *Proceedings of the 2013 International Conference on Software Engineering* (San Francisco, CA, USA) *(ICSE '13)*. IEEE Press, 802–811. https://doi.org/10.5555/2486788.2486893

[17] Serkan Kirbas, Etienne Windels, Olayori McBello, Kevin Kells, Matthew Pagano, Rafal Szalanski, Vesna Nowack, Emily Winter, Steve Counsell, David Bowes, Tracy Hall, Saemundur Haraldsson, and John Woodward. 2021. On the Introduction of Automatic Program Repair in Bloomberg. *IEEE Software* 38, 4 (2021), 43–51. https://doi.org/10.1109/MS.2021.3071086

[18] J. Kontio, L. Lehtola, and J. Bragge. 2004. Using the focus group method in software engineering: obtaining practitioner and user experiences. In *Proceedings. 2004 International Symposium on Empirical Software Engineering, 2004. ISESE '04*. 271–280. https://doi.org/10.1109/ISESE.2004.1334914

[19] Temur Kutsia, Jordi Levy, and Mateu Villaret. 2014. Anti-unification for Unranked Terms and Hedges. *Journal of Automated Reasoning* 52, 2 (2014), 155–190. https://doi.org/10.1007/s10817-013-9285-6

[20] Chen Liu, Jinqiu Yang, Lin Tan, and Munawar Hafiz. 2013. R2Fix: Automatically Generating Bug Fixes From Bug Reports. In *Proceedings of the International Conference on Software Testing, Verification and Validation (ICST)*. 282–291. https://doi.org/10.1109/ICST.2013.24.

[21] Martin Monperrus. 2014. A Critical Review of "Automatic Patch Generation Learned from Human-Written Patches": Essay on the Problem Statement and the Evaluation of Automatic Software Repair. In *International Conference on Software Engineering*. Association for Computing Machinery, New York, NY, USA, 234–242. https://doi.org/10.1145/2568225.2568324

[22] Martin Monperrus. 2019. Explainable Software Bot Contributions: Case Study of Automated Bug Fixes. In *Proceedings of the 1st International Workshop on Bots in Software Engineering* (Montreal, Quebec, Canada) *(BotSE '19)*. IEEE Press, 12–15. https://doi.org/10.1109/BotSE.2019.00010

[23] Yannic Noller, Ridwan Shariffdeen, Xiang Gao, and Abhik Roychoudhury. 2022. Trust Enhancement Issues in Program Repair. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 2228–2240. https://doi.org/10.1145/3510003.3510040

[24] Chris Parnin and Alessandro Orso. 2011. Are Automated Debugging Techniques Actually Helping Programmers?. In *Proceedings of the 2011 International Symposium on Software Testing and Analysis* (Toronto, Ontario, Canada) *(ISSTA '11)*. Association for Computing Machinery, New York, NY, USA, 199–209. https://doi.org/10.1145/2001420.2001445

[25] Viktoria Stray, Tor Erlend Fægri, and Nils Brede Moe. 2016. Exploring Norms in Agile Software Teams. In *Product-Focused Software Process Improvement*, Pekka Abrahamsson, Andreas Jedlitschka, Anh Nguyen Duc, Michael Felderer, Sousuke Amasaki, and Tommi Mikkonen (Eds.). Springer International Publishing, Cham, 458–467. https://doi.org/10.1007/978-3-319-49094-6_31

[26] Yida Tao, Jindae Kim, Sunghun Kim, and Chang Xu. 2014. Automatically Generated Patches As Debugging Aids: a Human Study. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 64–74. https://doi.org/10.1145/2635868.2635873

[27] Rijnard van Tonder and Claire Le Goues. 2019. Towards s/engineer/bot: Principles for Program Repair Bots. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*. 43–47. https://doi.org/10.1109/BotSE.2019.00019

[28] Emily Rowan Winter, Vesna Nowack, David Bowes, Steve Counsell, Tracy Hall, Saemundur O Haraldsson, and John Woodward. 2022. Let's Talk With Developers, Not About Developers: A Review of Automatic Program Repair Research. *IEEE Transactions on Software Engineering* (2022), 1–1. https://doi.org/10.1109/TSE.2022.3152089

[29] He Ye, Matias Martinez, and Martin Monperrus. 2021. Automated Patch Assessment for Program Repair at Scale. *Empirial Software Engineering* 26, 20 (2021), 1–38. https://doi.org/10.1007/s10664-020-09920-w

[30] J. Yi, U. Ahmed, A. Karkare, S. Tan, and A. Roychoudhury. 2017. A feasibility study of using automated program repair for introductory programming assignments. In *Proceedings of ESEC/FSE*. https://doi.org/10.1145/3106237.3106262