Information and Software Technology 190 (2026) 107932

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Newcomers' experiences during debugging: A cognitive inclusivity perspective using GenderMag[☆]

Faith Culas [*], Amisha Singh, Atharva Arankalle, Priyanka Dhopade, Kelly Blincoe

Human Aspects of Software Engineering Lab, Waipapa Taumata Rau | University of Auckland, New Zealand

ARTICLE INFO

Keywords:
Debugging
Cognitive styles
Software engineering
Diversity and inclusion
Software tools and practices

ABSTRACT

Context: Debugging is a critical practice in software engineering that enables software engineers to ensure the correctness of code by identifying and resolving bugs. It also benefits newcomers as it helps them go through the codebase, understand its structure, and learn about its functionality. Recent research has uncovered that some software engineering tools are not well suited to all ways of thinking, imposing an additional cognitive overhead on individuals whose cognitive styles are not well-supported by the tool. While biases have been explored in other software tools, little is known about whether debugging tools exhibit cognitive biases and introduce "inclusivity bugs".
Objective: This paper addresses this gap by examining inclusivity bugs that newcomers encounter when using the PyCharm debugger.
Methods: In this study, we performed a controlled lab experiment where we observed 24 software engineering students with little to no experience as they used the PyCharm debugger for a set of tasks. We used a think-aloud protocol to capture participants' thoughts throughout the experiment. Then, we conducted a thematic analysis, guided by our research question, to identify potential biases in the tool. We used the GenderMag framework to examine the relationship between cognitive style and the inclusivity bugs.
Results: We detail our findings on 21 inclusivity bugs which are caused by 2 main reasons: discoverability and learnability. We identified trends that showed individuals with low self-efficacy, low motivation, risk-averse tendencies, and those who prefer to learn by processes and gather information selectively were the ones who faced the most challenges.
Conclusion: The findings provide insights into how debuggers can be made more inclusive. They also highlight the need for continuous evaluation and adaptation of SE tools and practices to ensure they meet the needs of all users with diverse cognitive styles to ensure fairness.

1. Introduction

Software engineering (SE) is not just about writing code. It is a discipline that relies on various tools, practices, and standards to build software systems and to maintain them over time. Over the years, many software engineering tools and practices have evolved to assist software engineers in designing, building, maintaining, and improving software systems. There has been extensive research on SE tools and practices, focusing on identifying issues and recommending improvements to improve the way we engineer software (e.g., [1,2]). However, recent studies have uncovered a previously overlooked factor: cognitive biases in some widely used SE tools, such as GitHub [3–5] and Google's internal code review tool [6]. For example, a study of GitHub found barriers, including lack of visibility, lack of feedback, and information overload, that impacted people with certain cognitive styles [3]. Cognition is the "collection of mental processes and activities used in perceiving, remembering, thinking, and understanding, and the act of using those processes". Research from the field of psychology has shown people have various cognitive styles [7].

There are many cognitive processes involved in software engineering [8]. As a result, if the tools we use to engineer software are not cognitively inclusive, individuals with certain cognitive styles can experience barriers and incur a *"cognitive tax"*. In this context, *"inclusivity bugs"* refer to features of a tool that do not adequately support users with diverse cognitive styles. We adopt the same definition as Guizani et al. [5], that is, if groups of users ultimately complete their tasks

☆ This article is part of a Special issue entitled: 'Fairness in software systems' published in Information and Software Technology.
* Corresponding author.

E-mail addresses: fcul804@aucklanduni.ac.nz (F. Culas), asin448@aucklanduni.ac.nz (A. Singh), aara123@aucklanduni.ac.nz (A. Arankalle), priyanka.dhopade@auckland.ac.nz (P. Dhopade), k.blincoe@auckland.ac.nz (K. Blincoe).

https://doi.org/10.1016/j.infsof.2025.107932
Received 1 May 2025; Received in revised form 12 October 2025; Accepted 14 October 2025
Available online 23 October 2025

but face disproportionate barriers along the way, such as confusion, missteps, or the need for workarounds, those barriers are considered inclusivity bugs. In this paper, we refer to a bug in the code as "bug", while we refer to a problem impacting the inclusivity of the software tool as an "inclusivity bug" for clarity.

GenderMag [9] is a validated method to identify inclusivity bugs in software. It leverages personas that consider five different cognitive characteristics known as the GenderMag facets: *Motivation, Self-efficacy, Information processing style, Learning style, and Attitude towards Risk*. GenderMag has proven to be effective at identifying inclusivity bugs in software tools [4,10] including through large-scale industry studies [6]. One study on the GitHub interface [3] identified 12 inclusivity bugs across four common tasks done by newcomers. Prior research indicates that inclusivity bugs disproportionately affect certain cognitive styles and women [3–6,9]. While GenderMag has been applied to several software tools to find inclusivity bugs, most tools used in the software engineering process have not been studied from a cognitive inclusivity lens. There have been calls to action to investigate how to make software tools more inclusive [11].

In this study, we examine the inclusivity of a popular software debugger. Debugging is essential in software development for identifying and resolving bugs and issues. Debugging software is a complex task that can be time-consuming, often requiring more effort than creating the software [12]. Those new to debugging often find it difficult and frustrating [13]. Debugging tools (or debuggers) are integrated into popular development environments, such as Visual Studio Code and JetBrains' integrated development environments (IDEs). Debuggers allow software engineers to examine the state of a running software program. We examine the inclusivity of the PyCharm debugger through a think-aloud lab experiment to gather initial perceptions of the tools from newcomers (those with little or no debugging experience). PyCharm is a popular Python IDE developed by JetBrains with an integrated debugger. Our study was guided by the following research questions:

- RQ1: What inclusivity bugs do newcomers encounter when debugging with PyCharm?
- RQ2: How do the inclusivity bugs identified in the debugger relate to the GenderMag facets?

The results of the experiment revealed 21 inclusivity bugs related to 13 features of the debugger caused by two main issues: ***discoverability*** and ***learnability***. Discoverability issues are related to barriers in finding features within the debugger, and learnability issues are related to difficult to use features. The findings illustrate potential ways debugging can be made more inclusive.

2. Background

2.1. Cognitive styles

Systematic differences in how people process information, solve problems, and make decisions, referred to as cognitive styles, emerged as a research focus in the mid-20th century. Work done by psychologists like Herman Witkin [14], and Jerome Kagan [15], helped establish that these cognitive styles are not about ability but about preferred approaches to thinking. Research from subsequent decades revealed that cognitive styles are shaped by complex interactions of sociocultural context, educational experiences, and even gender. Gender research showed that socialization practices, not biological sex itself, influence whether individuals develop more comprehensive versus selective information processing styles, with studies by psychologists highlighting how stereotypes and expectations shape cognitive development from childhood [16–18].

2.2. GenderMag

GenderMag is a method that helps to identify inclusivity barriers in software products by considering cognitive style of potential users [19]. It was developed based on cognitive style research from the field of psychology that examined gender differences in problem-solving approaches. The method is based on personas namely, Abi, Pat, and Tim to portray differences in five cognitive facets: *Motivations* to use software, *Information Processing Styles*, *Learning Style*, *Computer Self-Efficacy*, and *Attitudes toward Risk* (see Table 1 for explanations of these different facets). Abi represents users with low self-efficacy, low motivation, who are risk-averse, have a process-oriented learning style, and prefer to gather information comprehensively before taking action. In the GenderMag method, evaluators conduct a cognitive walkthrough of a software product by considering how a persona would use the software. Abi is the most used persona as these traits often reveal inclusivity bugs since many software products are designed for users on the other end of the spectrum for each of these cognitive style facets. While GenderMag uses personas like Abi and Tim to represent two ends of the cognitive spectrum, it is important to note that cognitive styles are not binary. Neither is there a superior or better style; individuals lie in a continuum of different styles for each facet.

Using personas can be beneficial in helping users perceive a persona as a real person and empathize with the persona, but there are also limitations [20]. Particularly in the case of the GenderMag framework, personas can reinforce gender stereotypes, but the authors emphasize that personas were not designed to reinforce stereotypes but to compare differences among them based on the gender differences found in empirical studies. To further mitigate the effects of stereotyping, GenderMag uses gender-neutral persona names (e.g., Abi, short for either Abigail or Abishek) and provides multiple photos to represent each persona.[1] Using multiple photos instead of one promotes gender inclusiveness and at the same time does not reduce users' engagement with the persona [21]. GenderMag supports customization of personas as well [22]. In our study, we do not employ the GenderMag walkthrough, which involves role-playing using these personas. Instead, we use real participants to examine how people interact with the debugger in a lab experiment and use the GenderMag framework only as a lens for analysis.

3. Related work

3.1. Cognition in software engineering tools and practices

One lens that has been considered for improving software tools is cognitive inclusivity. Research examining cognitive inclusivity in software tools has been explored to see how users interact with tools such as GitHub [3,4], Stack Overflow [2], Visual Studio [9], development tools [22,23], and various custom tools like Google's internal code review tool [6]. These studies identified issues, or *"inclusivity bugs"*, that create barriers for individuals with diverse cognitive styles. One study found that software engineers spend 14% of their time fiddling with the UI of software tools, highlighting the need for more efficient UIs for software tools [24].

A study on GitHub [3] identified 12 inclusivity bugs which impacted the Abi persona. The GitHub interface was subsequently redesigned and evaluated using 75 newcomers, resulting in an increase in the completion rate of the experiment tasks from 67% to 97% for individuals whose cognitive styles were unsupported in the original design.

The Abi persona is often attributed to finding the most inclusivity bugs [25]. Although it may cause concern that fixing inclusivity issues from an Abi perspective could leave newcomers with non-Abi traits less supported, results indicate that the performance of all participants

[1] https://gendermag.org/foundations.php

Table 1
GenderMag cognitive facet description for each persona [19].

Facet	Persona description
Motivation	• Abi: uses technology only as needed for the task. Prefers familiar features • Pat: exhibits both Abi and Tim characteristics • Tim: uses technology to learn new features
Self-efficacy	• Abi: lower self-efficacy than their peers. Often blames self and might give up as a result. • Pat: medium self-efficacy. Keeps trying for a while when problems arise. • Tim: higher self-efficacy than peers. Usually blames technology when problems arise. Tries multiple ways before giving up.
Information processing style	• Abi and • Pat: Gather and read comprehensively before taking action. • Tim: Reads selectively. Follows any cues and backtracks.
Learning style	• Abi: process oriented • Pat: exhibits both Abi and Tim characteristics. Tries new features but does so mindfully. • Tim: tinkerer but this can be distracting
Attitude towards Risk	• Abi and • Pat: risk-averse. • Tim: risk-taker. Explores new features and enjoys doing so sometimes.

improved when inclusivity bugs are fixed [3]. This underscores the effectiveness of GenderMag and demonstrates how addressing inclusivity bugs can enhance performance for the entire population.

Furthermore, studies have shown that cognitive styles often cluster by gender, leading to inclusivity bugs that disproportionately affect women [2,4]. Burnett et al. [9] emphasized the importance of considering gender differences and thus cognitive differences when designing tools, but argued that such changes do not have to favor one gender at the expense of another. An effective example of achieving this is demonstrated by the work of Murphy-Hill et al. [6]. They found that the redesigned edit feature in Google's internal code review tool improved its discoverability for both men and women overall.

While some inclusivity bugs have been found in some of existing software engineering tools, very few tools have been studied from this lens. There is a need to study the cognitive inclusivity of more software engineering tools to improve their usability for all cognitive styles.

3.2. *Novice debugging*

Debugging is an essential practice that can help developers to read and understand code [26]. Novice software engineers often struggle with debugging [13,23]. Also, novices employ different debugging strategies and face greater difficulty compared to experienced software engineers [27]. In a debugging experiment [23], some participants skipped through the code or entire nested loops instead of stepping through the program carefully. This made it challenging to track variables, leading to confusion as they navigated the learning curve of understanding the debugger. Another study found that some students lacked confidence in their debugging skills and the fixes they applied [28]. Although it is well known that novices struggle with debugging, it is rarely taught or given limited emphasis in courses [29,30]. There has been research in improving debugging tools for newcomers such as implementing a reverse execution feature as debugging involves tracing back from the failure to identify its root cause [31].

Research examining debugging from a cognition perspective is fairly limited. An eye-tracking debugging study reported a strong correlation between debugging skills and cognitive activities [32]. Another research study on debugging strategies found gender differences in information gathering, with females being more comprehensive and males more selective before fixing bugs [33]. They explain this behavior using the selectivity hypothesis that predicts that women gather information comprehensively before acting upon it. While these findings point to different cognitive approaches to debugging, the role that debugging tools play in supporting or hindering remains unexplored, especially when considering the underlying cognitive differences between individuals. If there are cognitive biases incorporated in debuggers, this can introduce a *"cognitive tax"* that disproportionately affects certain cognitive styles, affecting debugging performances unfairly. Our study examines debuggers from a cognitive inclusivity perspective to fill this gap.

4. Methodology

To answer our research questions, we employ a think-aloud protocol [34] in a lab experiment, where participants performed six debugging tasks. The think-aloud protocol requires the participant to verbalize their thoughts so that the observer/interviewer can follow and understand the thought process of the participant. A qualitative approach was used to answer our research questions to explore the complexity of the problem in depth and further understand how participants interacted with the debugger.

In the following section, we explain our study design and analysis methodology. We first conducted a pilot study of all tasks (see Section 4.2 for details on pilot study), followed by the main experiment. The analysis was conducted in parallel with the experiments to ensure that data saturation was reached before concluding.

4.1. *Study design*

The think-aloud lab experiment consisted of a one-hour, individual session for each participant. The experiment session began with a short explanation of the goal of the study and the expected flow of the session. The session was divided into three sections: the warm-up, observation, and interview sections. Each section, described below, was semi-structured, with prompts to guide the discussion, but, based on the responses, additional questions were asked to gather more detail or insights into the participants' responses, allowing unforeseen information to be collected as well.

(1) Warm-up: The warm-up section was designed to take about five minutes to make the participant comfortable and settle in. In this section, we asked a few introductory questions, such as: "Are you enjoying your degree so far?", "What are your pronouns?", "What career do you aspire to go into after this degree?", and "What do you know about debugging code?".

(2) Observation: The main section is the observation where the participant was required to complete six debugging tasks using the PyCharm Community Edition IDE. The six tasks had to be completed in order. Each task involved one to two bugs inserted into a single python code file. The code used in these tasks was adapted from an prior code comprehension study [35]. This code was selected because it was validated by that prior study to be complex enough to be non-trivial, yet still understandable in under 30 min, ensuring participants would not become fatigued. The prior study was conducted at the same university (University of Auckland), so we drew from a similar participant pool. Bugs were manually inserted in the code for each task by the research team (see below Listing 1 for a small code snippet of one of the tasks).

```
# This function determines if the date that is inputted is a valid date or not
# Day, month and year are numeric inputs to the function
```

```
3 def is_valid_day_in_month(day, month, year):
4     month_length = LENGTH_OF_MONTH[month]
5     if month == FEBRUARY and is_leap_year(year):
6         month_length + 1
7     return day > 0 and day < month_length
```

Listing 1: Task 2 Python code

The bugs inserted reflected common errors made by novice software engineers, such as issues with loop counters and conditional operators, as identified in a three-decade literature review by Alzahrani et al. [36]. For example, in line 4 of Listing 1, we see a common error which is array indexing. Unit test cases were also added since using test cases can help participants both identify the cause of the bug and to verify that the program behaves correctly after the bug fixes. The six tasks were as follows:

Task 1: In the first task, the participants had to fix one small bug in a single line of code. The goal of Task 1 was for the participants to familiarize themselves with the IDE, the think-aloud protocol, and the experiment flow. They were not required to use the debugger to fix the bug.

Task 2: In the second task, which was slightly more complex than Task 1, the participants had to fix two bugs. While using the debugger would help the participants to observe the state of the program, no explicit instruction was given to use the debugger to find the bugs in this task. The goal of Task 2 was to enable us as researchers to observe each participant's debugging methodology and approach, without being given explicit instruction to use the debugger available in PyCharm. This allowed us to observe how they completed the task in a way that felt natural and comfortable to them.

Task 3: In the third task, the participants were introduced to the debugger in PyCharm, if they hadn't already used it. This task did not require the participants to fix any bugs in the code but rather allowed us to observe how successful each participant was in finding, running, and using the debugger to suspend the program and observe the state of the code at a particular point in execution.

Task 4: This task was a continuation of the same code in Task 3, with the goal of observing how successful each participant was in using basic debugger features (e.g., "step over"), examining variables, and fixing the bugs planted in the code. This task had two bugs to be fixed.

Task 5: The fifth task was designed to be the most challenging. Using the debugger, participants were required to find and fix two bugs planted in the code. The key difference in this task was that the bugs planted in the code were in a nested function. Therefore, we were interested in observing how successful each participant was in using more complex debugger features, such as "step in" and "step out".

Task 6: In the final task, each participant was prompted to explore the PyCharm debugger independently and further investigate any features they wanted to on their own. The purpose of the task was to observe how each participant used the debugger and its features without a specific end goal in mind, as well as to identify which features participants were drawn to and their reactions to these features.

(3) Interview: The third and final section of the experiment, the interview, was designed to enable the participant to reflect on the tasks and also to clarify things that happened during the observation. Open-ended questions were used (e.g., "How did you find the overall experience of debugging?", "What did you find the most challenging about debugging or using the debugger?").

Our replication package contains full details of the study design, including the python code used in the debugging tasks as well and the full list of interview questions [37].

4.2. Pilot study

Before recruiting participants, we conducted a pilot study to assess the feasibility of the experiment design and ensure that the tasks were both achievable within the given time and relevant to the research objectives. The pilot study included four participants, all of whom were final year undergraduate or first year postgraduate software engineering students with limited or no experience in debugging. These participants were recruited prior to sending out invitations to the wider networks for the actual study. The insights gained from the pilot study resulted in us modifying some aspects of the experiment design. The most significant changes were done to Task 1 and Task 4.

Task 1, although short and intended as an introductory task, was time-consuming for the pilot study participants due to the complexity of a conditional statement in the code used for the task. As a result, the code was refactored to improve clarity by simplifying the conditions. For Task 4, the change was to introduce an additional bug, as the original bug was too easy to identify, which contradicted the goal of the task.

The data from the pilot study was not used in our final analysis.

4.3. Ethical considerations

The study was approved by the University of Auckland Human Participants Ethics Committee (UAHPEC). Participants were provided with a Participant Information Sheet (PIS), which outlined the purpose of the study, how participants would be involved in the experiment, and their right to withdraw at any time. Participation was entirely voluntary.

Participants provided informed consent through a signed hard-copy consent form before taking part in the study. All information extracted from the sessions was de-identified, removing all personal details and stored without referencing any identifiable individuals. Each participant was assigned an ID (e.g. P1). Any potentially identifying information found during the anonymization process was kept confidential and accessible only to the research team.

Participants were given a $20 voucher as a token of appreciation for their time.

4.4. Participant recruitment

The study was advertised in university forums and using flyers among software engineering and computer science students, at the University of Auckland. Anyone interested in taking part in the experiment was asked to fill out a short questionnaire to express their interest and provide information about themselves. This questionnaire included questions to assess their cognitive style and questions about their gender and programming and debugging experience. Gender was collected since prior research has found that women are often disproportionately disadvantaged by cognitive biases in software tools [10].

To assess participants' cognitive style, we used the questions from the GenderMag facet questionnaire, which has been validated in prior research [38]. This includes 14 questions, each of which is answered using a 9-point Likert agreement scale. One of the statements in the questionnaire, for example, is *"I want to get things right the first time, so before I decide how to take action, I gather as much information as I can"*, which is used to assess the participant's information processing style.

A full copy of the recruitment questionnaire is available in our replication package [37].

4.5. Participants

From the 43 responses we received to the recruitment questionnaire, we invited 40 participants to participate in the experiment. The same invitation was distributed in three rounds through software engineering and computer science university forums and flyers. Each recruitment round lasted approximately one month, taking place between October 2024 and March 2025, with a break during the December–January university holiday period. We only selected participants with some programming experience but no or minimal debugging experience based on their questionnaire responses, as our focus was on newcomers encountering inclusivity bugs. Gender and cognitive styles were also considered for participant selection as we wanted a diverse range of cognitive styles across the participants.

The invited participants were contacted through email and were asked to book a suitable time for the experiment. Out of the 40 who were invited, 12 cancelled and did not schedule a time with us. Three participants did not show up despite having booked a time. One participant was excluded from the analysis because they demonstrated higher experience during the study. We included postgraduate students in our recruitment criteria as well. The University of Auckland has a diverse student body, with many international students, some of whom enter postgraduate study directly after completing a bachelor's degree. Also, the software engineering bachelor's degree at the University of Auckland is industry-focused. So even undergraduate students are taught debugging briefly. The one participant who demonstrated more extensive experience and was excluded from the analysis was actually an undergraduate. We only included students with little or no debugging experience. Postgraduate students also come from a wide range of backgrounds, such as computer systems or machine learning, often with minimal debugging experience. We also included two postgraduate students in our pilot study to assess whether postgraduate students would also be suitable participants.

In the first two rounds, due to a lower number of willing participants, we did invite three participants who had selected "I use them very often" when asked about their debugger experience. During analysis, we excluded one participant who clearly had prior debugger experience. The other two had rated their experience higher but appeared to overestimate their familiarity, still behaving like newcomers when using the debugger. In the last round of recruitment, we only invited participants who reported minimal or no experience with debuggers. Three participants were declined in the final round of data collection. Two were declined since they reported frequent use of the debugger. In addition, one participant, who identified as a man, was declined to improve gender diversity of the participant sample.

Among the 24 participants who were included in the final analysis, 10 identified as women, 13 as men, and one chose not to disclose their gender. Overall, while there were varying levels of programming experience, all participants included in the study had little to no experience in debugging. Four participants had never used Python but had experience with other languages such as Java, 11 used it sometimes, and nine used it very often. For debugging tools, eight had never used them, 14 used them sometimes, and the two who reported using them very often had minimal overall experience and were included in the analysis. Sixteen of the participants were undergraduate students and eight were either first or second year postgraduate students.

4.6. Study execution

The study was conducted in person in the Human Aspects of Software Engineering Lab at the University of Auckland. The sessions were individual, and the participant and observers were present in the same room. After receiving participant consent, data was recorded through audio and screen recording of the device used to complete the debugging tasks, allowing researchers to review the interactions the participants had with the PyCharm debugger. Sessions were held over the course of 3 months. There were a total of 24 sessions, with 20 h of recorded audio and screen recording.

The experiment sessions were conducted in three rounds with analysis occurring iteratively between each round. We stopped collecting data when no new inclusivity bugs were identified in a round and saturation was reached. During the first round, two undergraduate students jointly observed five participants. In subsequent rounds, participants were observed by a single observer (a Ph.D. student). To ensure consistency across rounds, the PhD student joined one of the sessions of the first round as a distant observer. The second round included 13 participants, and the third round included six participants. According to Guest et al. [39], in qualitative analysis, saturation typically occurs within the first twelve interviews, although basic elements for meta-themes can be observed as early as six interviews. We reached saturation after 18 experiments (the last experiment of the second round), when no new inclusivity bugs were identified. To confirm saturation, we conducted a third round of experiments in March 2025. No new inclusivity bugs emerged in the last round.

4.7. Data analysis methodology

To answer our first research question, we conducted a *reflexive* thematic analysis of experiment transcripts to identify inclusivity bugs in the debugger, which can be barriers to effective use of the tool. For the second research question, we analyzed the data through the lens of GenderMag to identify trends in cognitive styles disproportionately impacted by the tool's design.

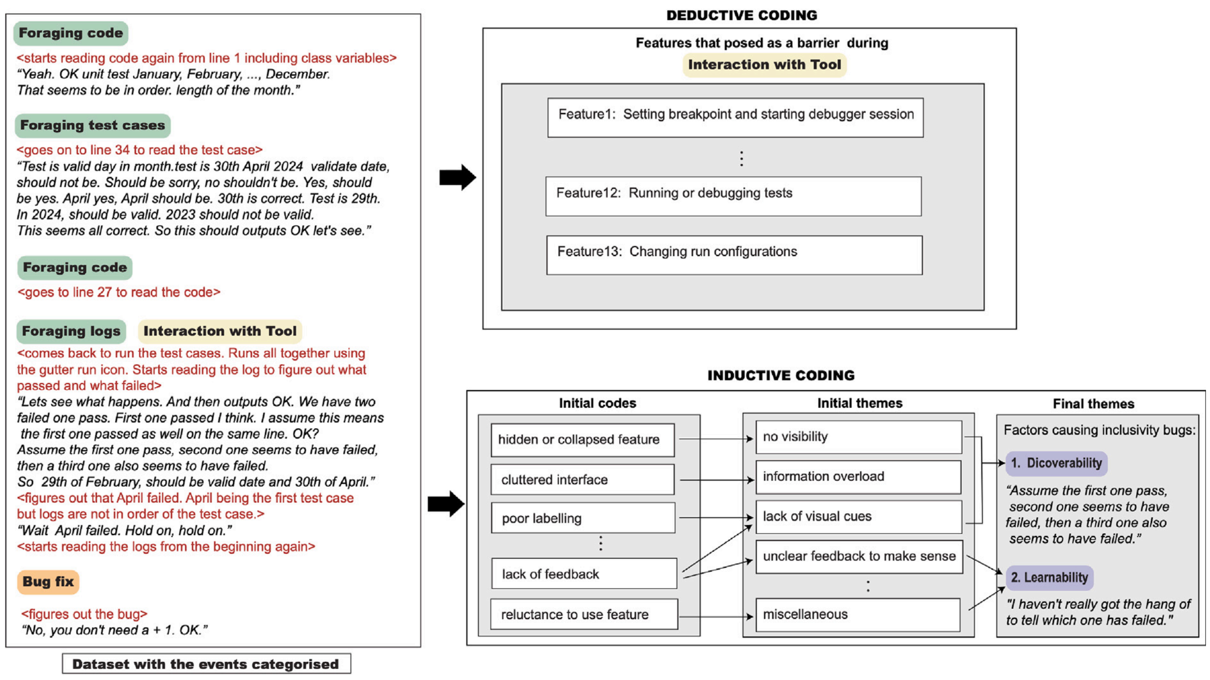
4.7.1. Reflexive thematic analysis

Since the initial publication by Braun et al. [40] on thematic analysis (TA), the term has evolved due to the nature of how qualitative analysis can be creative and flexible but also subjective to the knowledge of the researcher. So TA has now been revised and re-termed as *reflexive* TA. Our study is grounded in an interpretivist philosophical stance, recognizing that software engineering research involves understanding not just the technical aspects but also the human behavior that shapes software development processes. Thus, *reflexive* TA is well suited. We did not use code reliability measures because, consistent with the principles of *reflexive* thematic analysis, we view researcher subjectivity as a valuable resource in knowledge production rather than a bias to be minimized, and coding reliability metrics are meaningless with this epistemological stance [41].

First, we generated transcripts from the recorded sessions. The transcripts were generated by listening to the audio recordings alongside the screen recordings, capturing both the verbal responses and actions of the participants. For example: <starts reading code again from line 1 including class variables> *"Yeah. OK unit test January, February, …, December. That seems to be in order. length of the month."* Actions were recorded inside <>. This was necessary because the experiment focused on participants' interactions with the user interface, and some insights could only be fully understood by linking verbal responses to on-screen actions. To ensure the quality of the transcripts, the recordings were watched at least twice.

Next, we created a relevant data set from the generated transcripts. This was done by applying the sensemaking model for end-user debugging [33] adapted from Pirolli et al. [42]. The model highlights three sensemaking loops: the bug fixing sensemaking loop, environment sensemaking loop, and the common sense/domain sensemaking loop. The model also includes a sub-loop known as the foraging (information gathering) sub-loop. We applied all components of the model except the domain sensemaking loop, as it focuses on reasoning that involves recalling domain-specific knowledge. Since the tasks for the lab experiment involved basic leap year calculations and did not require domain knowledge, this loop was not relevant in this context.

Using the model, when preparing the data set, we categorized events in the session timeline as *bug fixes*, *interactions with the environment/tool*, and *foraging* aligning with the relevant sensemaking loops. There were a large number of foraging events, so we further categorized these events based on where participants searched for information: foraging code, foraging test cases, foraging logs, foraging using debugger, and foraging documentation. For example, when participants read lines of code, including code comments, the activity was recorded as "foraging code". When participants tried to understand the code using the debugger, the activity was recorded as "foraging using debugger". We included only the level of detail necessary for our analysis in the extracted data set; for instance, P18 after her bug fix ran the test cases and was surprised all test cases passed as she assumed she had to


Foraging code
<starts reading code again from line 1 including class variables>
"Yeah. OK unit test January, February, ..., December.
That seems to be in order. length of the month."

Foraging test cases
<goes on to line 34 to read the test case>
"Test is valid day in month.test is 30th April 2024 validate date, should not be. Should be sorry, no shouldn't be. Yes, should be yes. April yes, April should be. 30th is correct. Test is 29th. In 2024, should be valid. 2023 should not be valid.
This seems all correct. So this should outputs OK let's see."

Foraging code
<goes to line 27 to read the code>

Foraging logs Interaction with Tool
<comes back to run the test cases. Runs all together using the gutter run icon. Starts reading the log to figure out what passed and what failed>
"Lets see what happens. And then outputs OK. We have two failed one pass. First one passed I think. I assume this means the first one passed as well on the same line. OK?
Assume the first one pass, second one seems to have failed, then a third one also seems to have failed.
So 29th of February, should be valid date and 30th of April."
<figures out that April failed. April being the first test case but logs are not in order of the test case.>
"Wait April failed. Hold on, hold on."
<starts reading the logs from the beginning again>

Bug fix
<figures out the bug>
"No, you don't need a + 1. OK."

"Assume the first one pass, second one seems to have failed, then a third one also seems to have failed."

"I haven't really got the hang of to tell which one has failed."

Dataset with the events categorised
DEDUCTIVE CODING
Features that posed as a barrier during
Interaction with Tool
Feature1: Setting breakpoint and starting debugger session
⋮
Feature12: Running or debugging tests
Feature13: Changing run configurations
INDUCTIVE CODING
Initial codes
hidden or collapsed feature
cluttered interface
poor labelling
⋮
lack of feedback
reluctance to use feature
Initial themes
no visibility
information overload
lack of visual cues
unclear feedback to make sense
⋮
miscellaneous
Final themes
Factors causing inclusivity bugs:
1. Dicoverability
2. Learnability

Fig. 1. Thematic analysis process (both inductive and deductive).

fix two bugs. She spent 6 min evaluating the test cases, even though her understanding of the code was correct and all test cases passed. This was simply recorded as *<P18: sounds surprised as all test cases pass.>"That because I thought I need to do 2 fixes, but it fixed both test cases. Now I'm going to check whether that's correct or not?"<participant checks two test cases individually going through the code linearly by mentally calculating and then says>"Peace of Mind. OK. OK. OK."* All activities in the transcript were categorized into events as bug fixes, interactions with the tool, or one of the types of foraging before moving onto further analysis. See the left panel of Fig. 1 for an example of a transcript with the events categorized.

After generating the transcripts and extracting the relevant dataset, we used thematic analysis to analyze our data, following the six-phase process outlined by Braun and Clarke [40].

(1) Familiarizing yourself with your data: During transcript generation described above, the coder (the first author) watched all recordings at least twice. She also read and re-read the extracted data set to become familiar with the data.

(2) Generating initial codes: When generating the initial codes we used a combination of both deductive (research question driven) and inductive (data driven) approaches.

The deductive codes were related to the features of the debugger. The barriers encountered during *interaction with the tool* event were iteratively coded into 13 distinct features of the PyCharm debugger driven by our first research question focusing on interactions with the debugger. Participants interacted with several debugger features, such as adding breakpoints, starting the debugger, viewing variables, and examining test results.

Alongside with the deductive codes, we also applied an inductive approach to understand why certain features became barriers. When coding we considered both semantic and latent levels. In thematic analysis, semantic level codes capture the explicit content of the data and the latent level codes go beyond what is said to interpret underlying meanings. For example, when looking for the debugger start button, P1 reached out for help and when shown how to, says *"Ah I just don't see that"*. This was eventually coded into "hidden in plain sight feature". To explain an example from a latent level, P20 confused about why the wrong task was running said: *"You know what I was confused about. If I go to task three and four and I start playing, why is it still playing task five? Is it just always gonna save the previous one?"* This was due to the participant not knowing that the run configurations had changed. This was interpreted as "lack of feedback from tool" due to no response from the tool indicating the user that run configurations have changed.

In our study, the first author coded the extracted data set. The coder primarily relied on transcripts but occasionally re-watched participant videos to clarify specific moments when additional context was needed. Rather than treating the researcher's perspective as a bias to eliminate, *reflexive* TA acknowledges that their experiences, interests, and professional background can provide valuable insights into the data. To support transparency, we have also included a positionality statement (see Section 4.7.4).

(3) Searching for themes: In this phase, the initial codes were sorted and grouped into higher-level initial themes. Since we had both semantic and latent codes, we searched for both explicit and interpretive themes.

For example, the scenario mentioned above, the code "hidden in plain sight" contributed to the initial theme of "lack of visual cues". Codes "poor labelling" and "lack of feedback from tool" also contributed to the initial theme of "lack of visual cues". The code "lack of feedback from tool" also contributed to another theme " unclear feedback to make sense" if the lack of feedback made it hard to make sense of what the feature does but not in finding the feature (see the right bottom panel of Fig. 1).

(4) Reviewing themes: During this phase, the themes were reviewed and, based on their relevance to the research question, some were merged, split, or discarded. For example, "lack of visual cues", "information overload", "no visibility", all were coded into one high level theme — discoverability. This was done through iterative discussions between the authors.

(5) Defining and naming themes: All themes were defined with descriptions of their scope and content. The names were selected to capture the core essence of each theme. All authors reviewed the theme names and their descriptions.

(6) Producing the report: The final phase involved selecting example quotes for each theme and explaining the themes in the manuscript. Again, this was reviewed by all authors.

Table 2
Number of participants across each facet tagged as Abi-like, Pat-like, and Tim-like. For the Pat-like participants, we show a further breakdown showing where they fall in the quartiles.

Facet	[min-Q1] Abi	(Q1-Q2) Pat (towards Abi)	Q2 Pat	(Q2-Q3) Pat (towards Tim)	[Q3-max] Tim	Total no. of participants
Motivation	6	5	3	0	10	
Self-efficacy	6	5	2	2	9	
Information processing style	7	4	4	3	6	24
Learning style	6	4	5	1	8	
Attitude towards risk	6	6	0	6	6	
Total	31	24	14	12	39	

Table 3
Examples of participant actions and verbal utterances showing how features were coded as inclusivity bugs using the definition from Guizani et al. [5].

Reason for classifying as inclusivity bug	Participant actions and verbal utterances
Confusion	P12: <participant goes line by line using debugger>…“I’m not sure what just happened”. <Continues clicking step into my code and goes into next test case…still uses only step into my code. Gets confused which test case as still runs whole test suite.> “I’m not entirely sure how the debugger works”
Missteps	P14: “Ah I see what you mean, I think there is watch” <starts right clicking on the line with breakpoint and looks for watch…. Participant fixated on finding the variables using the watch feature of the debugger as he had heard of it before but wasn’t aware that he couldn’t see the variables because the debugger was not running.>
Workarounds	P13: “Is this testing for 1904?”…. <uses print statement to see the outputs of the boolean logic. Prints year also to make sure which test case ran. Checks logs with the print statements to make sense of what the outputs are.>
No progress	P8: <introduced the stepping buttons at Task5 as participant continues only with resume and it’s very hard to follow through variables>“these ones?” <participant chooses to use step-over>… “Yeah. Yeah. So I think that did help as well.”
Asks for help	P6: <clicks debugger but no progress as no breakpoints. Puts breakpoint but keeps clicking the run button this time as debugger didn’t make sense last time. Looks for the debugger. Gives up after looking for it. Goes back to reading the code.> “can you show me how to use debug in Pycharm and see the variables?”

Table 4
Number of participants who encountered inclusivity bugs categorized by debugger feature and cause (discoverability and learnability). - indicates no inclusivity bug.

	Feature description	Discoverability bug	Learnability bug
1	Setting breakpoint and starting debugger session	5	2
2	Finding the debugger icon to start the debugger	3	-
3	Stopping debugger session	1	-
4	Setting breakpoint at the correct line	-	7
5	Following the execution point	1	2
6	Stepping through program	3	21
7	Examining variables	4	4
8	Managing breakpoints in the middle of a debug session	-	7
9	Evaluating expressions	6	7
10	Resuming program	4	5
11	Exploring test results	13	1
12	Running or debugging tests	14	-
13	Changing run configurations	6	3
	Total instances of inclusivity bugs	60	59

4.7.2. GenderMag analysis

To identify the cognitive style of each participant, we used their responses to the GenderMag facet questionnaire to calculate a score for each of the five facets [38]. After reverse scoring any negatively worded questions, the scores are summed for each facet. While the original scoring method allocated anyone over the median as Tim-like and anyone under the median as Abi-like [38], we choose to also include Pat-like to have a more detailed view of the facets. For each facet, participants were labeled as Abi-like if their score was less than or equal to the first quartile (25%) and Tim-like if their score was greater than or equal to the third quartile (75%). Those whose scores fell between the first and third quartiles were labeled as Pat-like. The Pat-like category reflects a range of traits, as participants may exhibit a mix of Abi-like and Tim-like tendencies. For clarity, we described these participants as leaning “towards Abi” or “towards Tim,” without allocating them fully to either Abi or Tim (see Table 2).

Even though GenderMag brought to life three facets, Abi, Pat and Tim, they also remind the users that these facet values lie in a continuous scale and there are a range of facets that can be considered [38]. While Pats have a unique mix of both Abi and Tim traits, for 2 facets, *information processing style* and *risk attitude*, they align with Abi [19]. Keeping this in mind, in our analysis, we have not assigned Pat randomly to either Abi or Tim, but rather presented the facet scores as it is, on the spectrum of each of the facets. This approach allowed us to make a finer-grained comparison across all three personas and to include Pat in the results. The scores are not absolute but rather a comparison of the facets within the sample.

To analyze the relationship between the identified inclusivity bugs and the GenderMag facets, we plotted the distribution of participants’ facets scores and visually compared the participants who experienced the most number of inclusivity bugs (greater than or equal to the third quartile in terms or number of bugs encountered) and the least number of inclusivity bugs (less than or equal to the first quartile). The GenderMag facet scores were normalized for the plots to improve readability. We looked to see if those who experienced the most bugs tended to cluster into any of the personas for each facet (e.g., Abi).

With relation to interpretation of the GenderMag facets with the participant behavior, we would like to highlight that we did not code facets to our data. After identifying the inclusivity bugs in the debugger using TA, we examined each of the five facet scores separately to enable a deeper understanding of the relationship between cognitive style and inclusivity bugs. Then we backtracked into our transcripts to find illustrative quotes and behavior that show the relationship between facet tendencies and the inclusivity bug. When looking for matching patterns in our data we used exemplar quotes to ground our interpretations from previous GenderMag studies [5,10]. For example, *"...this leads me to a page with the bare minimum of instructions... I have no idea where to go from here"*, demonstrates Abi-like learning style.

However, as the facets cannot be fully separated, the selected quotes do not indicate that only a single facet was involved in a particular inclusivity bug. We discuss more about the interweaving between the facets in Section 5.2.2. We also would like to highlight that inline with the goal of the experiment, we were not interested to see if participants fix all bugs but rather how they use the debugger to fix the bugs and the barriers associated with it.

4.7.3. Operational definition of an inclusivity bug

In our study, we use the definition of an inclusivity bug from Guizani et al. [5]. If users ultimately complete their tasks but face disproportionate barriers along the way, such as confusion, missteps, or the need for workarounds, those barriers are considered inclusivity bugs. For example, P20 immediately noticed that the run configurations had changed and quickly adapted by finding a workaround; however, it was not a barrier to carrying on with the tasks.

> *P20: "OK, so play it again. Oh." <run configurations are still at task1. Participant notices that its still running first task> "Oh, oh, I must play this. I have no doubt this works. I'm gonna guess yes. OK." <takes a go at a workaround running the play from the gutter> "So let's pass the 1st others failed. Actually. Oh, it's not in order. OK, it's passed the last one, but not these two. I see. OK".*

But for P4 this feature of changing configurations became a barrier.

> *P4: "I don't see the other 2 test cases." <participant still not got the hang of the configuration changing.> "but it says passed 0. I think it passed".*

Examples illustrating each type of barrier in our study are shown in Table 3.

4.7.4. Positionality statement

When using *reflexive* TA, the research team's experiences and professional backgrounds become tools for interpreting the data, so it is important to acknowledge our position. The research team comprised of three students and two experienced researchers. The first author, a doctoral student, is an experienced software engineer with industry experience across three companies in two countries, and has first-hand knowledge of the challenges of using software engineering tools working across multiple teams. The two experienced researchers, both affiliated with the university, bring expertise in research and teaching in engineering. The other two team members were final-year undergraduate students from the University of Auckland.

5. Results

Before presenting the findings to answer our research questions, we first give an overview of the participants' task performance and debugging choices. Even though most of the participants were able to find all of the bugs, only just over 70% were able to fix them all. This was due to the first task, which contained a bug in a conditional statement with boolean expressions (see line 5 of Listing 2). Excluding Task 1, nearly 85% of the participants were able to fix the remaining bugs.

```python
1 def is_leap_year(year):
2     if year % 4 != 0:
3         return False
4
5     if year % 100 == 0 or year % 400 != 0:
6         return False
7
8     return True
```

Listing 2: Task 1 Python code

5.1. Identifying inclusivity bugs with the PyCharm debugger

We identified 21 inclusivity bugs (see Table 4) related to 13 features of the debugger that pose as barriers for newcomers. From our thematic analysis, we identified two main factors causing the inclusivity bugs: Discoverability and Learnability.

5.1.1. Factors causing the inclusivity bugs

First, we explain these two main factors causing the inclusivity bugs. Then, we explain how they manifested in each of the 13 features.

1. **Discoverability:** The degree to which users can independently locate features or functionality without requiring external assistance. There are many factors that can reduce discoverability. Some of the factors that were found in this study were information overload due to cluttered interface, suboptimal placement, lack of visual cues, hidden or collapsed elements. Poorly chosen names or the absence of descriptive labels can make it difficult to understand the purpose of features and thus leading to non-discoverability.
2. **Learnability:** The degree to which users can understand and use a new feature. Insufficient or unclear feedback can slow down the sense-making process. Sense-making is a key part of learning, as users need to understand the tool's responses and form appropriate goals and next steps to effectively navigate and understand what features do.

Each feature with inclusivity bugs had either a discoverability bug, a learnability bug, or both.

5.1.2. Inclusivity bugs in PyCharm

In this section, we describe the inclusivity bugs impacting each of the 13 features. Recall that even if participants ultimately find and make use of the feature, we still classify it as an inclusivity bug if they encountered barriers along the way.

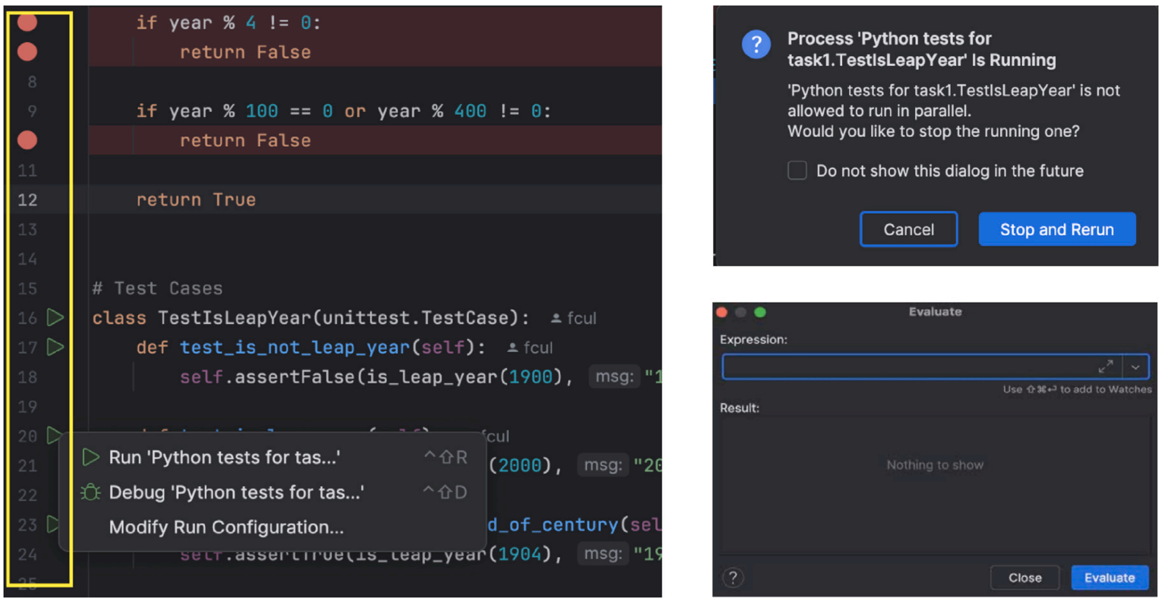
Feature 1: Setting breakpoint and starting debugger session

This feature had two inclusivity bugs: discoverability and learnability. In Task 3, participants were guided to suspend the program at a particular line. However, nearly a third of them (seven participants) found this challenging.

Discoverability: Five participants had difficulty figuring out how to set a breakpoint to suspend the program at a specific line. One participant consulted the documentation and, lots of tinkering, managed to place the breakpoint. Others could not figure it out and either asked for help or were given help when they could not progress further. Breakpoints must be placed on the gutter, which is located on the left of the code editor (see Fig. 2). The gutter also contains the run button and the line numbers appear unclickable, which may have contributed to their difficulty.

> *P16: <participant tries clicking on the gutter line numbers but breakpoints do not get inserted.> "I need to put more points. How to put? Using this?" <clicks view breakpoints button as an alternative>*

Learnability: A key step to starting the debugger and suspending the program is by setting breakpoints first. Understanding how to use



```
    if year % 4 != 0:
        return False

    if year % 100 == 0 or year % 400 != 0:
        return False

    return True



# Test Cases
class TestIsLeapYear(unittest.TestCase):  fcul
    def test_is_not_leap_year(self):  fcul
        self.assertFalse(is_leap_year(1900),  msg: "1
                                      fcul
                                      (2000),  msg: "20
                                      d_of_century(sel
        self.assertTrue(is_leap_year(1904),  msg: "19
```

Process 'Python tests for task1.TestIsLeapYear' Is Running

'Python tests for task1.TestIsLeapYear' is not allowed to run in parallel.
Would you like to stop the running one?

Run 'Python tests for tas...' ^⇧R
Debug 'Python tests for tas...' ^⇧D
Modify Run Configuration...
Do not show this dialog in the future
Cancel
Stop and Rerun
Evaluate
Expression:
Use ⇧⌘⏎ to add to Watches
Result:
Nothing to show
Close
Evaluate

Fig. 2. The gutter area highlighted in yellow with the debug icon hidden in gutter. It needs to be right clicked to open debug option (see left), Pop-up dialog in Pycharm when stopping current session (see top-right) and Evaluate expression dialog (see bottom-right).

the breakpoint before starting the debugger and what it does was a barrier for two participants. In Task 3, when participants were asked to suspend the program at a specific line, P8 successfully places the breakpoint by clicking on the line and then mentions: *"I am not completely sure what it does. Does it like remove the line?"*. He successfully finds the debugger icon but removes the breakpoint before starting the debugger and was not sure whether the debugger had started. P16, despite receiving help and successfully setting her first breakpoint, continued to struggle with setting breakpoints throughout the experiment. She repeatedly used the manage Breakpoints window, which, while allows breakpoint to be added, also contains several other options that can overwhelm a newcomer

> *P16: <Even after interviewer helped participant with placing breakpoint but again when placing next breakpoint>"I want to put a breakpoint in line 12 but its very difficult for me"*

She struggled to use the manage breakpoints window to add new breakpoints and needed multiple attempts and assistance to fully grasp the feature.

Feature 2: Finding the debugger icon to start the debugger

This feature posed as a challenge due to not being able to be discovered easily. Only three of the participants found this as a barrier.

Discoverability: The small debugger icon being in the right top corner and also hidden collapsed in the gutter/test runner tab makes it hard to be seen (see Fig. 2). P1 received a hint to help her as she was unable locate the debug icon and said, *"Ah, I just don't see that"*. Another participant, P14 set the breakpoint but ran the tests without the debugger (using the run button and not the debug button) and assumed the debugger started looking at the expected and actual value printed in the test logs. He was eventually provided a hint as well to try running the test cases differently.

> *P14: "Is what I am looking at the debugger. I am expecting one thing and its giving something else. My immediate thought would be to follow the calculations that lead to these numbers. How does it get there? You told me run the debugger and then I found these issues and then I trace it back where and I run the debugger there and from there till I find something that works." <interviewer asked participant to examine state of the code at specified line> "Ah I see what you mean, I think there is watch" <starts right clicking on the line with breakpoint and looks for watch>*

Feature 3: Stopping debugger session

Only one participant encountered this discoverability inclusivity bug.

Discoverability: After successfully setting the breakpoint and starting the debugger, one participant, did not know to stop the debugger. He tried to start the debugger again without stopping the previous debugging session, which resulted in a pop-up window appearing prompting him to stop and rerun the debugger (Fig. 2). He was confused and hesitant to click on the stop and rerun button.

Feature 4: Setting breakpoints at the correct line

This feature was identified as a barrier due to a single inclusivity bug: learnability.

Learnability: After figuring out how to set the breakpoint, figuring out where to set it was a challenge for nearly a third (seven) of the participants. Breakpoints must be set on any executable line of code and not the method signature. Some participants set the breakpoint in the method signature, and, as this does not suspend the program, they struggled to progress with the debugger. Some participants set the breakpoint at the return statement line and were confused as to why they cannot observe the other variables. Four participants struggled but were eventually able to make sense of it, but three of them had to be guided to change the breakpoint to the line that needed to be examined. P5, one of the participants who eventually placed the breakpoint correctly at the desired location, after a bit of a struggle, expressed self-doubt, saying *"I am not sure whether I have put the breakpoints correctly or not"*.

Feature 5: Following the execution point

This feature exhibited two inclusivity bugs.

Discoverability: Only one participant, P8, after successfully starting the debugger doesn't realize what happened and reached out for help saying: *"Can you explained to me what happened?"*. He does not realize the code has been suspended at the highlighted line of code, and he does not see the variable tab in the debug tool window.

Learnability: Two participants who noticed the blue highlighted execution point, tried to make sense of it. They were confused as to whether the line was already execute or not. But gave up and eventually asked for help.

> *P1:"does it stop before this line or after this line?"*

Feature 6: Stepping through the program

A very important step needed to navigate through the code using the debugger while also having control of the execution of the code are the stepping functions of the debugger. The stepping buttons are located on the debug window toolbar. This feature exhibited both inclusivity bugs, and only three (12%) of the 24 participants were able to find and make sense of the step-over button, making it a significant barrier.

Discoverability: Finding the stepping buttons was a barrier to three participants. P1 reached out for help to find the stepping buttons and was pointed towards the stepping buttons. Even after showing the stepping buttons, P1, when looking to navigate into the nested function, hovers over all the stepping buttons but does not click on them as she does not realize what the labels mean. She later commented if "step into" had been named "step into function" it would have been more helpful. Another participant, P20, was able to discover the stepping buttons easily but mentioned that the stepping icons are so small and it would be better if the order of them were changed. He said, the step-into-my-code is more important than step-into. P4 on the other hand wanted to use the debugger during Task 2 and says *"I want to step through, to see line by line but I am not sure how to do that"*. She goes on to click around for some time and eventually sees the stepping buttons with no assistance due to her tinkering but when clicked on the step-over, gets confused as it takes her into library classes.

> *P4: "I think I did step-over and it took me somewhere else. I don't know which one I want to use..... I assume I step over now. I am scared it will take me somewhere else"*

P8, being one of the three who found it hard to find the stepping buttons, clicks only the resume button to navigate the code. Due to the participant's failure to see the stepping button, completing the task became more challenging as it was hard for him to keep track of the variables as the resume button does not go through each line of the code. However, as he was continuing the tasks only with the resume button, he was eventually given a hint at Task 5 pointing towards the stepping buttons.

Learnability: This bug was the most frequently encountered, affecting 21 participants. One participant, P1, while stepping through the code mentioned: *"I feel like it will be easier for people who know how to use it, but I don't"* and abandoned the stepping functions and resorted to adding and removing breakpoints and restarting the debugging session every single time a new breakpoint was needed. Understanding the differences and purposes of each stepping button was not very intuitive.

> *P18: "Step over and step into what's the difference between these two? I don't know what's the difference between this two. OK and step out is going back. But if I want to go back to my code only I can click that blue arrow thing." <points step-into-my-code> "And I'm bit confused with step over and step into."*

Feature 7: Examining variables

The Variables pane of the Threads&Variables tab of the PyCharm debugger window allows the user to examine the values stored in the objects of the program. This feature was both a discoverability and learnability inclusivity bug.

Discoverability: Finding the variables tab was challenging task for four of the participants. To highlight a few scenarios, one participant, P19 reached out for the documentation for help but still struggled saying *"I'm still looking for threads and variables tab"* and eventually was given a hint to look at the bottom debug window. Another, P4 was looking for the variables in the right place, but as the wrong thread was selected in the frame stack, the correct objects were not showing up. The Frames pane within the variables tab shows all threads of the application. Since she had selected the wrong thread, she needed a lot of tinkering to be able to locate the current thread. Another, P10, on the other hand gave up quickly and asked for help saying *"sorry not able to find"*.

Learnability: Four others, even though discovered the variables pane easily, found it uncomfortable to use and preferred printing the values on the console.

> *P9: "Is end month starting from 0 or 1? I'm low key tempted to console log stuff"*

Feature 8: Managing breakpoints in the middle of a debug session

This feature had a single cause for being identified as an inclusivity bug, which was its barrier in learnability.

Learnability: It was not intuitive that breakpoints could be added or removed during the debugger session without having to restart the session. This became a disadvantage to seven of the participants as they kept restarting the debug session which was time consuming and hard to follow through the code execution.

Feature 9: Evaluating expressions

This feature of the debugger helps to evaluate expressions or code fragments.

Discoverability: In our study, this feature would be handy to evaluate in the case where multiple conditions are in a single expression but out of the 24 participants, only five explicitly asked how to evaluate expressions, looked for the feature, and found it. The feature was not intuitive. 13 participants didn't look for it at all and continued with their tasks. The six participants who found this barrier didn't use the term "evaluate expressions," but their workaround with the use of print statements or splitting up conditions suggested they were trying to inspect or check expressions.

Learnability: Even the five participants who found it, couldn't make sense of how to use it. P19 mentioned, *"it looks like a calculator. Don't know what it does"*, and moves on. P7 mentioned, *"evaluate expression. my guess it evaluates it without going into the line"* and then clicks on it and an empty window opens and says *"oh alright, that doesn't do anything"* (see Fig. 2). When explained what it does, participants mentioned that its one of the most useful features in the debugger but its not intuitive.

Feature 10: Resuming program

When the debugger is suspended at a breakpoint, the resume button allows the program to continue running until the next break point (if any) or until the program terminates, eliminating the need to step through each line manually. This feature was found as a bug under both causes.

Discoverability: Three participants specifically asked about resuming but did not succeed in finding the feature. P7 spent time looking for it, and while they were able to find the button, it did not work as they anticipated since the breakpoint was not placed correctly, so they ultimately gave up saying:

> *P7: "how do I run till the next breakpoint? is it resume. Nope that ran the whole thing. I don't want to step through all 3 lines of the loop. I know its possible, but how do I do that?" <the breakpoint must be in the first line of loop to continue to next loop. But participant has placed breakpoint one line above it and clicking resume finishes the program. So participant searches for 'run until next breakpoint' but documentation overwhelming. Looks for continue for some time and gives up eventually as he wants to progress with the task> "OK i guess I'm gonna, ill just step through every single line."*

Even though he found the feature, he did not realize that is what the feature is for as the breakpoint was placed incorrectly, thus making the feature not discoverable. One participant who looked for the continue/resume button gives up looking for it and finds it later when continuing with the tasks, another participant was stuck in other library classes for quite some time and had to be pointed out to resume the program even though they did not ask for the feature.

Learnability: Out of the participants who found it, only two made sense of what the feature does. The rest (five participants) either misinterpreted its function or did not know how to use it and were reluctant to use the feature. For example, P9 struggled to understand the button's functionality after using it and expressed risk-averse feelings, stating *"I don't trust resume program any more. I don't know what it does. So I'm gonna step over"*. P20 misunderstood the functionality of the resume button as skipping over the loops.

Feature 11: Exploring test results

The Test Runner tab opens in the Run tool window when a testing/debugging session starts. The left pane of the tab shows the tree view of test cases along with their statuses including whether they have failed or passed. This feature had both discoverability and learnability barriers.

Discoverability: Find the test runner tab was a barrier for just over half (13) of the participants. Participants who faced this barrier spent time trying to understand the logs and figuring out what test cases failed and the reason for the failure. The test results in the test runner tab is right next to the console and error logs, but due to the cluttered interface was difficult to find. Two participants even made guesses *P2: "I assume it's the first one that passed"* and carried on with the task. P18 was relieved that she didn't have to check which test cases failed, *"OK. Oh, all three test cases failed. Oh, yes. Three failed. OK, I don't need to check."* as she had not figured out that the test results can be easily explored using the test runner tab. When asked why she did not see the test runner tab, her response was, *"maybe because it was hidden like this and it was not very visible. Maybe I saw it, but I I didn't mind to explore."*

Learnability: Only one participant who found the test runner tab found it difficult to understand what it means. P4, who found the feature easily, struggled to understand what the icons and statuses on the test cases represented.

> *P4: "I haven't really got the hang of to tell which one has failed."*

Feature 12: Running or debugging tests

Test cases can be run or debugged individually or as a whole. This was a feature which was hard to be found by 14 (58%) of the participants. The individual run icons for each test case are located on the gutter of the IDE. The default Python3 unit test behavior is that test cases are run in an arbitrary order unless specified otherwise. Also to debug them individually, either the gutter run/test runner tab icon next to the desired test case must be expanded (see Fig. 2) revealing the debug icon or the run configurations must be changed (this was classified as another barrier - Feature 13)

Discoverability: Figuring out how to control the execution of each test case was beneficial for those who discovered this feature. P9 mentioned, *"I don't want it to run like the whole test suite"* and looks how to run test cases individually but doesn't find the debug in the gutter but tinkers around and opts to right click the test case and chooses debug. Those who faced this barrier found workarounds to keep track of which test case was being debugged by following variables and printing variables but it was not easy to follow if breakpoints were not placed in the right places (as some breakpoints are not reachable for some tests).

> *P15: "I have no idea what I am or how I am debugging the tests."*

> *P4: "I want the first test case to fail. I don't know why its going to the last test case. It has already passed"*

Feature 13: Changing run configurations

This had both a discoverability bug and learnability bug.

Discoverability: Six of the participants encountered issues with the run configurations being changed in PyCharm. They were unaware that the configurations had changed and were confused why the expected testcases were not being run. One of them who faced this barrier figured out an alternate way to run the desired file after some tinkering by right clicking on the file and running it directly. The other five were given hints due to confusion about what happened.

> *P24: "No, at the beginning there was like a screen which basically told me everything I passed and everything I failed. So you want to see everything that yes, do I need to unclick these?" <asks should she remove the breakpoints. participant hasn't realised run configurations have changed and only one test case running>*

Learnability: Three participants had difficulty in using the feature. P9, realized that the configurations changed, and correctly clicks on the run configuration option but was not clear on how to change it back to the current file being tested.

> *P9: "am I allowed to figure out here?" <clicks on the run configurations> "its nothing important here"*

Two other participants had to be repeatedly reminded to change the run configurations. Although they knew where to make the change (so not a discoverability bug), they were often confused during the course of the experiment because they did not realize the configuration had changed. This was mainly due to lack of feedback from the tool that the run configurations had been changed.

RQ1

What inclusivity bugs do newcomers encounter when debugging with PyCharm?

Answer: We identified 21 inclusivity bugs related to 13 features of the debugger that newcomers encounter when debugging with Pycharm caused by two reasons: discoverability and learnability.

5.2. Analyzing relationship between the identified inclusivity bugs and the GenderMag facets

We answered our second research question by comparing GenderMag facet scores with the likelihood of facing inclusivity bugs. We do this considering first the overall number of inclusivity bugs participants faced, followed by looking at the number of inclusivity bugs broken down by the two main factors of discoverability and learnability. Fig. 3 shows the participants who encountered the least and most inclusivity bugs using the quartiles as the cut-offs. Figs. 4 and 5 are the distribution of participants who encountered inclusivity bugs for specific causes (learnability and discoverability respectively).

5.2.1. Overall - All inclusivity bugs

Fig. 3 shows the participants who faced the most number of inclusivity bugs (in black) and those who faced the least number of inclusivity bugs (in white). As described in Section 4.7.2, the first and third quartiles were used as cutoffs for identifying the participants who faced the least and most inclusivity bugs. When considering all of the inclusivity bugs, the cut-offs were three or less for those facing the least number of inclusivity bugs and seven or more for those facing the most inclusivity bugs.

As illustrated in Fig. 3, most participants who encountered the highest number of inclusivity bugs are clustered around the Abi and Pat personas (with Pats leaning towards Abi) for four facets: motivation, self-efficacy, learning style, and risk attitude, and around the Tim and Pat personas (with Pats leaning towards Tim) for the information processing style facet. This suggests that the inclusivity bugs were more strongly associated with Abi-like and Pat-like participants. However, this was not uniform. Participants faced less issues were not limited to a single facet type but were observed across the spectrum, highlighting

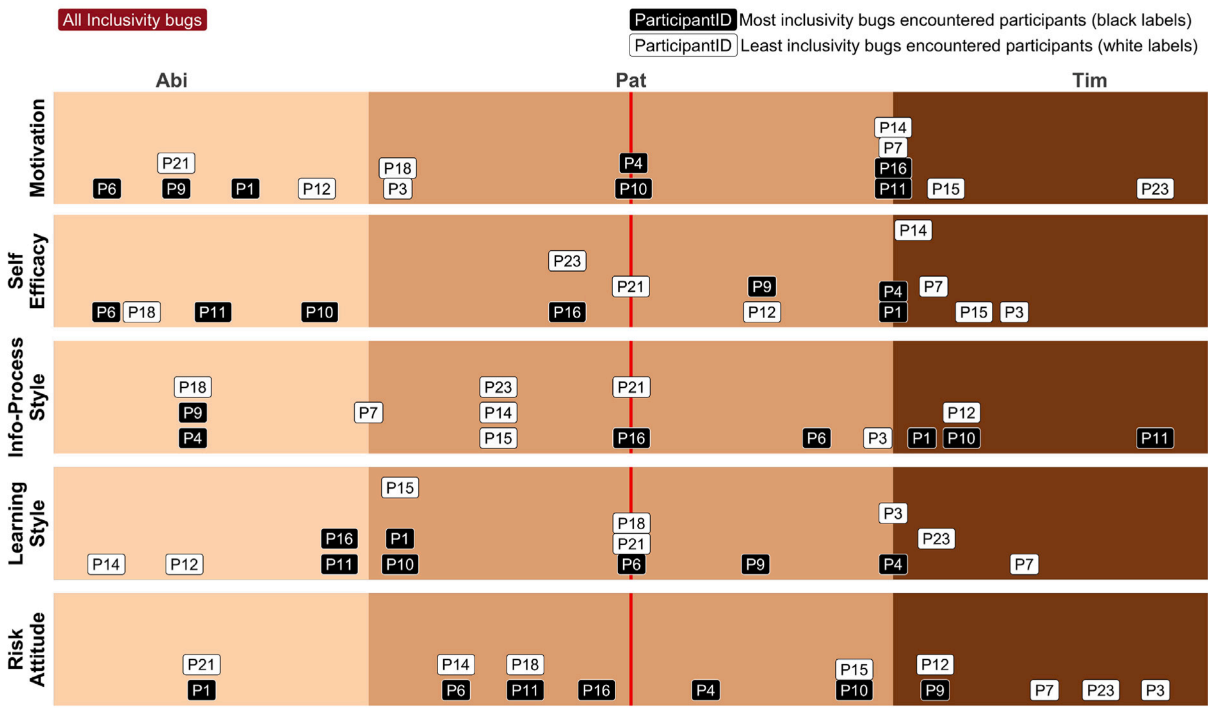


Fig. 3. Facet distribution of all participants with most (in black labels) and least (in white labels) inclusivity bugs **(both causes)** encountered highlighted as Abi, Pat, **Tim** (Here most and least are categorized using **Q1 and Q3** as the cut-off. The red line is the median of the facet distribution.).

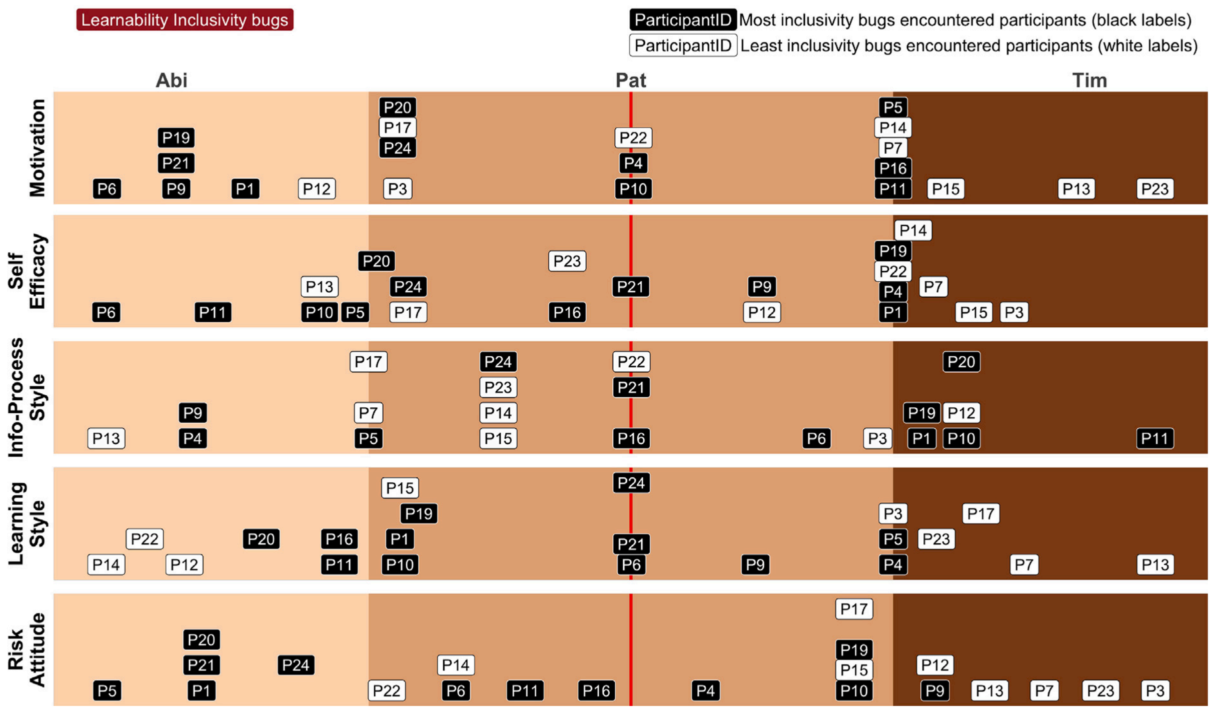


Fig. 4. Facet distribution of all participants with most (in black labels) and least (in white labels) inclusivity bugs **(learnability only)** encountered highlighted as Abi, Pat, **Tim** (Here most and least are categorized using **Q1 and Q3** as the cut-off. The red line is the median of the facet distribution.).

individual variation and the interweaving of multiple facets which we discuss more in detail in Section 5.2.2.

To investigate deeper into how the facets related to the inclusivity bugs, we next examine these trends by considering the two main causes of inclusivity bugs, discoverability and learnability. We present our findings on learnability first due to a larger number of participants falling below the first quartile and above the third quartile, allowing for greater differentiation between those who faced the most and least inclusivity bugs for this factor.

5.2.2. *Learnability bugs*

The number of learnability inclusivity bugs faced ranged from zero to six. Fig. 4 shows the participants who faced the most and least learnability bugs. The most, shown in black, faced three or more (third quartile), while the least, shown in white, faced one or less (first quartile). There are 21 participants included in this graph based on these cutoffs.

Twelve participants faced three or more learnability bugs. Of those, we see that for four facets, except information processing style, Abis and Pats encountered more inclusivity bugs than Tims(see Fig. 4). In the motivation facet, 42% of participants are Abis, 33% are Pats, and 25% are Tims; among the Pats, half lean toward the Abi end of the spectrum. In the self-efficacy facet, 33% are Abis, 42% are Pats, and 25% are Tims, with 60% of the Pats leaning toward Abi. For learning style, 25% of participants are Abis, 58% are Pats, and 17% are Tims; among the Pats, three toward the Abi and one toward Tim. For risk attitude, 42% of participants are Abis, 50% are Pats, and only 1 participant is a Tim. The only facet where Tim personas encountered more issues than Abi personas was information processing style, with five participants compared to Abi's three. We observe that white labels (participants who faced the fewest inclusivity bugs) tend to cluster towards the Tims end for facets such as motivation, self-efficacy, and risk attitude. In contrast, black labels (participants who faced the most inclusivity bugs) cluster towards the Abis end for facets such as motivation, self-efficacy, learning style and risk attitude. These are just trends we see with a small sample, so we refrain from providing any quantitative results for a general population.

To better understand the learnability bugs in relation to cognitive style, we examine them based on the five GenderMag facets as explained in the methodology in Section 4.7.2. Here we present our results in detail under each facet with illustrative quotes and actions. Because individual quotes do not always map neatly onto a single facet as seen in previous studies as well [5], throughout this section we draw on multiple quotes to illustrate the broader patterns we observed between facet scores and the inclusivity bugs. Quotes are thus used to support the trends rather than as definitive evidence of a facet in isolation.

Motivation: As seen in Fig. 4, out of the 12 participants who faced the most learnability inclusivity bugs, five lie in the Abi region, four lie in the Pat region and three in the Tim region. Individuals with low motivation tend to stick to familiar features and avoid exploring additional ones. However, failing to find or understand additional features can also make using the other features more difficult. For example, participants who did not discover how to run or debug individual test cases found it more difficult to follow through the state of the code as participants were confused as to which test case was being run, especially if breakpoints were not in optimum positions. We also observed different levels of motivation, with some participants not even trying to look for features and others spending significant time attempting to find features before giving up and reverting to familiar features. For instance, when looking for the resume button, P1 did not try to look for it further.

> *P1: "so this is one pass of the for loop, can I get it to do a second pass?" <participant was given a hint to look at the tool window but does not look for the resume function and continues with task>*

But P9 and P7 spent some time looking for it and ultimately gave up.

> *P9: "I want to see the loops its going through. I wish there was a continue or something" <participant looks for resume and gives up.....> "I don't even know what I'm debugging.... I'm low key tempted to console log stuff."*

> *P7: "I don't want to step through all 3 lines of the loop. I know its possible, but how do I do that". <Keeps looking for the resume button and then says> "I'll just step through every single line"*

Self-efficacy: We see a similar trend for self-efficacy as well. Of the 12 participants who faced the most learnability inclusivity bugs, four are in Abi region, five Pat and three Tim. Abi and Pat usually display low to medium self-efficacy. We also observed participants with high self-efficacy facing more inclusivity bugs, such as P1, P4, and P19. One possible interpretation is that, for example, P4, despite scoring high on self-efficacy, her risk attitude is low (Note that Abi-like and Pat-like participants share the same traits for risk-attitude - See Table 1). She was reluctant to try to new features to learn what they do. When making sense of what the stepping buttons so, she said, "I assume I step over now. I am scared it will take me somewhere else".

If we take the feature, setting breakpoints, six of them had difficulty in making sense where to set the breakpoints, and they were eventually guided to change the breakpoint to the line that needed to be examined (even though they knew how to place the breakpoint, they struggled learning how to use it and placing them in optimum lines). For example, even after guiding P5 to place breakpoints at an executable line of code, they continued to express self-doubt:

> *P5: "I am not sure whether I have put the breakpoints correctly or not"*

Some participants (for e.g. P3) were very confident of their actions and their bug fixes indicating high confidence and self-efficacy:

> *P3:"pretty sure it will work."*

Learning style: Here too we see similar trends with 10 of the participants facing the most learnability bugs lying in the Abi and Pat region (three Abis, seven Pats and two Tims). For example, it was not intuitive that breakpoints could be added during the debugger session without having to restart the session. This became a disadvantage for process-oriented learners with low self-efficacy and low tolerance for risk, leading to less tinkering, such as P10, P11, P16, P20 and P21, as they kept restarting the debug session which was time consuming and hard to follow through the code execution. When participants got confused, they tend to be reluctant to use the features again as well.

> *P16: "I need to check days" <changes break point to 30 and restarts debugger.> "I need to restart yeah".*

But despite being at the lower end of Abi for learning style, some were still able to overcome barriers. This highlights the role of multiple facets, which we discuss further in the section on interweaving facets (see Section 5.2.2).

Risk-attitude: This was the most significant barrier where almost all of the participants who faced the most learnability inclusivity bugs lie in the Abi and Pat regions, who tend to be risk-averse. For instance, learning how to set the breakpoint at the correct line as desired to follow the program execution was a challenge for seven participants. All seven participants were risk-averse (aligned with either Abi or Pat).

Another feature which disadvantaged risk-averse participants was the stepping buttons. Those who found it or needed help to find it abandoned it due to reluctance to use it as they got confused of what they actually do. Out of the participants who were reluctant to use the feature (P1, P4, P17, P20, P8, P12, P9 and P14), we see that most of them (except P8, P12 and P9) are Abis and Pats, who tend to be risk-averse. While trying to get the hang of the stepping buttons, P4 expressed fear despite being Pat-like leaning towards Tim's end.

> *P4: "I assume I step over now. I am scared it will take me somewhere else"*

P12, although scoring higher than P4 on risk tolerance, was reluctant to try new features since the step-into-my-code function in their code was already working fine. This could be due to low motivation and a preference for familiar features, but using just that stepping button eventually led to confusion.

> *P12: <sticks to using only one stepping button. Reluctant to try others>"I'm not entirely sure how the debugger works"*

Resuming program is another features that out of the five who faced learnability issues, four of them were risk-averse.

> *P9: "I don't trust resume program any more. I don't know what it does. So I'm gonna step-over to see"*

Information-processing style: Seven out of 12 who faced the most inclusivity bugs were Abi and Pat like (recall that Abi-like and Pat-like participants share the same traits for information processing style — see Table 1). Notably, almost all Tim-like participants for this facet encountered the highest number of inclusivity bugs, with five out of six affected. So, this was the only facet where most of the Tim-like participants encountered the most learnability inclusivity bugs. Being selective with information can be a disadvantage, as not all relevant information may be known. For example, during Task 5, P10 said:

> *P10: "what is number of days in the test case?"*

He was unable to make sense of the debugging process as he had not examined the test cases.

Inter-weaving of facets: While our analysis above considered each facet individually, the facets can influence each other. For example, while having a low motivation to try new features was seen as a disadvantage for many, some participants overcame and did not face many learnability bugs despite having low motivation. This was likely due to their other facets. For example, P3 aligned with Tim on all facets except motivation and did not face any learnability bugs. P12, who also scored low on motivation, demonstrated a comprehensive information-processing style, even though their score categorized them as selective. They thoroughly read all the code and test cases before starting the tasks and consulted the debugger documentation, which helped them better understand the debugger features throughout the tasks. P17 too overcame their low motivation because of their tendency to tinker. We also see some of the participants who encountered the fewest learnability inclusivity bugs (P13, P17, and P23) aligning with Abi and Pat in self efficacy. They appear at the tinkering end and risk-tolerant which may have been an advantage for them despite their lower self efficacy.

Conversely, being high in a particular facet does not mean you will not face inclusivity bugs. For example, P19, P4 and P1 scored high self-efficacy, yet still faced many learnability bugs. They all often reached out for help and were reluctant to use features despite their high self-efficacy. This may have been due to their risk-averseness. P14, who was fixated in finding the variables using the watch feature of the debugger got distracted of the main task in hand due to his high motivation to find the feature. We also see that other participants who faced more inclusivity bugs like P5 who scored a Tim in learning-style, Abi in self-efficacy and low tolerance to risk, again highlighting the interplay of facets.

Comparing P9, who encountered five learnability inclusivity bugs, and P12, who encountered only one, both showed reluctance to engage with features that caused confusion. Despite their risk tolerance, they tended to rely on familiar features.

> *P9:"I don't trust resume program any more. I don't know what it does. So I'm gonna step-over to see"*

> *P12: <looks at documentation and finds the step into my code. Continues using that for the rest of the task5>*

They exhibited low to medium self-efficacy and motivation, which may have amplified their reluctance to continue using features that were confusing, leading them to stick to familiar features instead. Risk aversion and low self-efficacy are related and can amplify each other. For instance, risk-averse people experimenting with new debugging features may feel discouraged if their initial attempts are unsuccessful. This can further lead to reduction in self-efficacy, making the perceived risks of adopting these features evengreater [33].

It is interesting to note that none of our participants fell cleanly into one single persona for all five facets.

5.2.3. Discoverability bugs

The number of discoverability inclusivity bugs faced ranged from zero to five. However, many of the participants fell between the first and third quartiles. Considering only those who faced the most discoverability bugs (scores ≥ third quartile threshold of 4) and the least discoverability bugs (scores ≤ first quartile threshold of 1), resulted in only 10 participants being considered for this analysis. The GenderMag facet plot of all participants who faced the most and least discoverability inclusivity bugs can be seen in Fig. 5. Again, participants who faced the most are shown in black and those who faced the least are shown in white. However, due to the small sample size, it is difficult to identify trends at each facet level. Instead, we examine individually the participants who faced the most and least discoverability inclusivity bugs.

(1) *Participants who faced the most inclusivity bugs*: The participants who struggled the most to find features in the debugger were P1, P4, P10, P16 and P17.

P1 is someone with low motivation, who is risk-averse, and has a more process-oriented learning style (low tinkering). She has high self-efficacy and has a more selective information processing style. Despite her high self-efficacy, she often reached out for help during the tasks, which could have been due to her attitude towards risk and low motivation to try unknown features. Her selective information processing style also disadvantaged in her in seeing things. For instance, she was unable to locate the debug icon, the resume icon, and the stepping buttons. She needed hints from the observer to find all of them. When shown the debug icon, P1 says, *"Ah I just don't see that"*, which could have been due to her selective processing, compounded together with her reluctance to tinker around, risk averseness, and low motivation. Her process-oriented learning style was evident when she hovered over the stepping buttons, trying to learn more about what they would do, but did not click on them. Her risk aversion further disadvantaged her in this situation. She became overwhelmed when looking at the test logs and said *"oh boy"*, but she was not motivating to look for another way (i.e. test runner tab) to explore test results.

P4 is also someone with high self-efficacy, but with somewhat higher motivation than P1. She is a tinkerer but gathers information comprehensively and is somewhat risk-averse. She was able to find some features that P1 could not find, like the stepping buttons and the test runner tab after a bit of tinkering. However, she had difficulty in finding the variables pane as her tinkering led her into the wrong frame. While she was very thorough when observing what happened when she was using the debugger, she expressed confusion when desired test cases weren't running. She was not motivated to find a way to run test cases individually.

> *P4: "I want the first test case to fail. I don't know why its going to the last test case. It has already passed"*

P16 had high motivation, but had low self-efficacy, was risk-adverse, and had a process-oriented learning style. She had difficulty in figuring out how to start the debugger session itself. She was motivated to figure out how to start the debugger and consulted the documentation, but struggled to add a breakpoint on the gutter. Her process-oriented learning style might have disadvantaged here as there was no clear instruction how to add breakpoints and the gutter seemed unclickable. When clicking on the gutter didn't work, she continued to be motivated and looked for another alternative way by clicking the "view breakpoints" window. Despite her high motivation to look for alternative methods, her risk averseness and low tinkering eventually led her to express difficulty and reach out for help.

> *P16: "I want to put a breakpoint in line 12 but its very difficult for me"*

P17 had low motivation and low self-efficacy, and was very comprehensive in gathering information. He was a tinkerer and was also risk averse. When looking for the variables tab, he could not find it

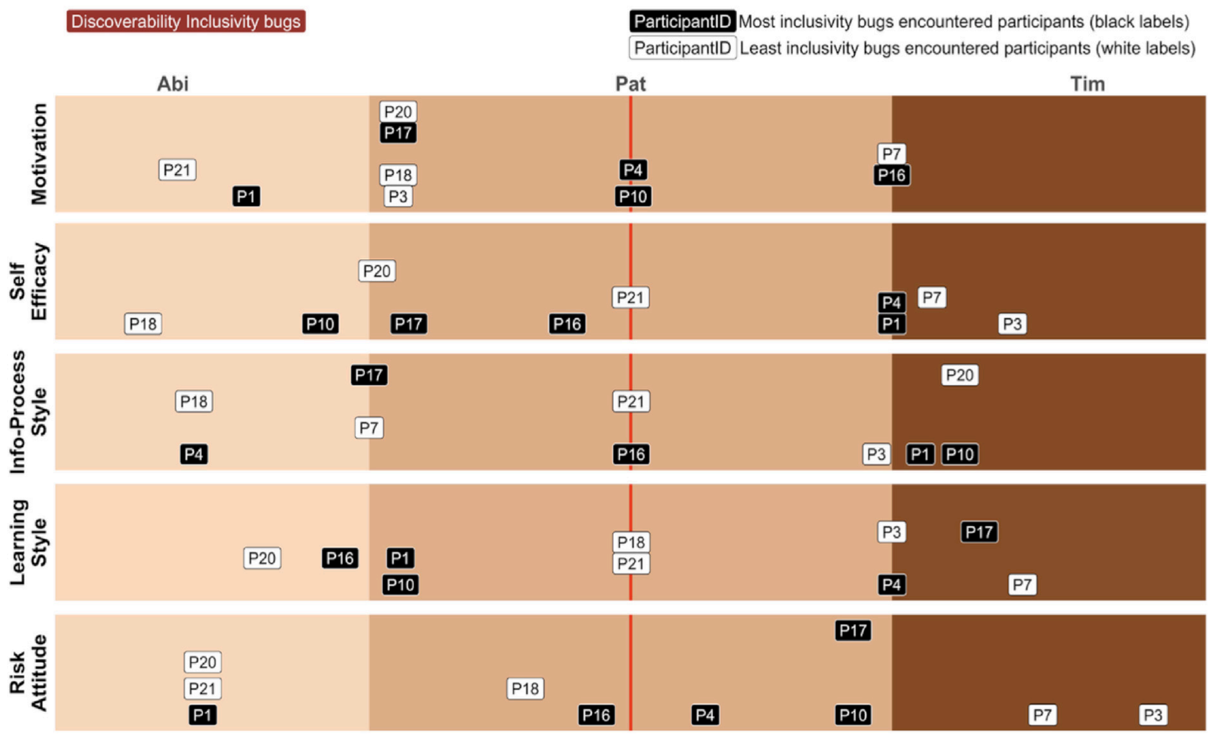


Fig. 5. Facet distribution of all participants with most (in black labels) and least (in white labels) inclusivity bugs **(discoverability only)** encountered highlighted as Abi, **Pat**, **Tim** (Here most and least are categorized using **Q1 and Q3** as the cut-off. The red line is the median of the facet distribution.).

and eventually gave up and needed help. His low motivation and low-self-efficacy might have contributed to this. He kept apologizing again and again as well.

> *P17: <keeps looking for variables in the logs and gives up.> Just let me see. I see it. Oh, maybe I have no see very useful information. I didn't see the useful information.*

P10 also gave up when looking for the variables pane. He has medium motivation, low self efficacy, is process oriented, more risk tolerant (a Pat but more towards Tim), and very selective. Even though he is risk-tolerant, his other four facets (more Abi and Pat like) could have disadvantaged him.

> *P10: <starts looking for the variables pane. Clicks some buttons in the debug tool window but doesn't see variables pane>. "sorry not able to find."*

(2) *Participants who faced the least inclusivity bugs*: The participants who encountered the least discoverability inclusivity bugs were P3, P7, P18, P20 and P21.

Both P3 and P21 faced zero discoverability bugs. While P3 had little debugger experience, P21 had never used them before. P3 is more selective in processing information, but tinkers and has a high tolerance to risk. They were willing to click around in the debugger no matter what the outcome was. P21, even though he was a Abi or a Pat for all facets, carefully tinkered throughout the experiment and was able to find the required features for the tasks.

P18 had similar personas as P21, but also exhibited tinkering throughout the experiment. She was also very observant when run configurations changed and was able to fix them successfully. She was not able to find the test runner tab but found how to run test cases individually to overcome that barrier.

> *P18: "I couldn't clearly find which test case fails OK. In when I run whole code. OK OK, so I'm now debugging 1 by 1, OK running test case. This is 1 by 1"*

P7 scored a Tim for all facets except information processing style. P7 only had difficulty in finding the resume button. He was motivated to keep looking for the resume for some time, and was tinkering around the debugger, but eventually gave up looking for it to continue with the tasks.

> *P7: "I don't want to step through all 3 lines of the loop. I know its possible, but how do I do that". <Keeps looking for the resume button and then says> "I'll just step through every single line"*

Even though P20 is more towards Abi for all facets except information gathering, he faced only one discoverability inclusivity bug (finding evaluate expression).

> *P20: <steps into and comes to main function line 28> "But this is right, so that's fine." <participant doesn't sound so sure if it the function returns true and tries to find a way to see what the helper function returns. Eventually asks for help> "Is there a way you can know what this is returning?"*

With the exception of that one feature, he was very intuitive in finding all needed features even though he had never used debuggers before.

Overall, we found that tinkering was most useful for avoiding Discoverability bugs. Even participants who were Pat-like tinkered. Low self-efficacy and risk-averseness, compounded with selective information processing, contributed to more discoverability bugs.

RQ2

How do the inclusivity bugs identified in the debugger relate to the GenderMag facets?

Answer: We identified trends for the learnability bugs, which showed that Abi and Pat were the most disadvantaged with respect to Self-efficacy, Learning Style, Motivation, and Risk attitude. With respect to Information Processing Style, it was Tim who encountered the most learnability bugs. Discoverability bugs were spread more evenly across the facets. Tinkering helped reduce discoverability bugs.

6. Discussion

In this section we interpret the results, discuss the implications of our findings for researchers and practitioners, explain the limitations and threats to validity of the study, and discuss potential future work.

6.1. Consistency with prior studies

Our findings align with prior studies on applying GenderMag to SE tools, reaffirming that Abi encounters more challenges. In our study, we did not focus solely on Abi and Tim; we also examined the Pat persona. The findings suggest that participants who shared characteristics with Abi and Pat in terms of self-efficacy, learning style, motivation, and risk attitude were most likely to encounter inclusivity bugs, indicating that the tool often fails to support Abi-like traits. Additionally, participants with a Tim-like information processing style also experienced a high number of inclusivity bugs, highlighting that even some traits associated with the Tim persona are not well supported by the tool.

Similar to prior research, our participants often had a combination of facets and did not align strictly to a single GenderMag persona [5, 10]. By analyzing the facets individually, we showed how different combinations of facets can either disadvantage or benefit users while using the debugger. For example, both P1 and P4 encountered difficulty in finding the stepping buttons. P1, who had low self-efficacy a process-oriented learning style (both Abi traits), was not able to find the buttons on her own. However, P4, who had a tinkering learning style (a Tim trait), clicked around for some time and eventually found the stepping buttons with no assistance. This illustrates the need for the debugger to be modified to suit different cognitive styles more equitably.

Past research has shown that women tend to skew towards Abi and men towards Tim, thus leading to gender bias [6,10,43]. Vorvoreanu et al. [10] conducted a qualitative study with 20 participants on the use of search engine with and without GenderMag treatment. Eleven identified as women and nine as men. Their results show that in the original version, women had failure rates twice that as men and after the GenderMag treatment on the software, the gender gap was eliminated completely. Murphy-Hill et al. [6] conducted a large scale investigation of GenderMag at Google on the code review tool. They report data from over 30,000 users. They too found that before the GenderMag redesign, it took women more time to discover the feature than men. But after the redesign with GenderMag, the gender gap disappeared. We did not see this clear difference between genders in our 24 participants. However, out of the seven participants who faced the most inclusivity bugs, five of them were women, while out of the eight who faced the least, only two were women. While these findings are based on a small sample and not statistically generalizable, they align with trends reported in prior research on gender bias in software tools. Our results reinforce the importance of considering inclusivity in the design and evaluation of software engineering tools and practices to suit all ways of thinking.

6.2. Limitations of GenderMag

While GenderMag is a useful framework and has proven effective in identifying inclusivity bugs, it has limitations, which we discuss here.

The scope of inclusivity is gender focused: GenderMag, which is grounded in empirical research on gender-related cognitive differences, focuses solely on one dimension — gender. This limitation has been addressed through introducing InclusiveMag, which generalizes GenderMag to handle multiple dimensions of diversity beyond gender [44]. The InclusiveMag process can be used to develop evaluations specific to other dimensions of diversity, such as age, socio-cultural aspects, or ethnicity. Our study focuses on gender differences, but future work exploring additional dimensions of diversity can build upon InclusiveMag.

Binary or limited gender representation: Although GenderMag aims to make software more gender inclusive, the research on which it is grounded is primarily focused on binary gender only. GenderMag has acknowledged limitations in its binary and limited gender representation, primarily focusing on “masculine” and “feminine” cognitive style clusters rather than capturing the full spectrum of gender identities. Much of the underlying psychological research, including that which informs GenderMag, has similarly been constrained by limited gender representation. However, recent studies have begun to address this gap. For example, a study extending GenderMag by including transgender and non-binary personas found that gender and appearance do influence the GenderMag evaluation process [45].

Lab-based experiments, such as ours, are not limited to binary gender if participants are diverse across the gender spectrum, but the analysis is still constrained by the GenderMag framework that was based on binary gender. Research in cognition has found that transgender individuals often exhibit cognitive patterns that align with their gender identity rather than assigned sex [46]. Joel et al. [47] demonstrate that most individuals possess a “mosaic” of masculine and feminine cognitive traits rather than purely binary patterns. Morgenroth et al. [48] propose to move beyond gender binaries in cognitive research entirely, a perspective that challenges frameworks like GenderMag to evolve toward more intersectional, identity-inclusive approaches and to update their personas in line with new cognition research.

Persona perception and stereotyping effects: Currently, GenderMag uses multiple photos to represent each persona as a way to mitigate stereotyping without reducing engagement with the persona. However, research shows that when personas have no specific gender is assigned, users often default to assuming the persona is male [25]. We addressed this concern by using actual participants to identify inclusivity bugs rather than employing the cognitive walkthrough with predetermined personas. However, we acknowledge that categorizing participants into three personas for analysis inevitably leads to some oversimplification and cannot fully capture the nuances of individual differences [20].

Using the median to categorize Abis and Tims: In the GenderMag method, facets are assigned based on the median. Scores to the right of the median are Tim-like and scores to the left of the median are Abi-like. This median-based approach, however, overlooks nuances among participants whose scores fall near the boundary between personas. To address this limitation of GenderMag and to add more nuance to personas across the spectrum, we included the first quartile and third quartile boundaries in addition to the median. This way, we can differentiate Abi-like Pats and also Tim-like Pats. However, we observed a considerable proportion of participants clustering on or near the quartile boundaries, including the median. This highlights that, while personas provide a useful heuristic, categorizing individuals into discrete groups risks overlooking subtle but meaningful differences among users whose characteristics fall adjacent on a spectrum. We identify the treatment of individuals on quartile boundaries as a key area for future research, particularly in developing methods that better account for users at the edges of persona categories.

Facet mapping with in-situation behavior was not always exact: Although participants were assigned to a persona based on their questionnaire scores, their behavior during the experiments did not always align with their assigned persona classification. While GenderMag has validated its questionnaire, other studies have also reported imperfect alignment, with only 78% of participants' in-situ facet verbalizations corresponding to their questionnaire responses [5]. This behavior could be due to several factors. The experimental setting itself may alter participants' natural behavior. Moreover, the GenderMag persona scores are not objective but rather relative to the sample. GenderMag was originally validated on a more general population of software users, where there may be more diversity across cognitive styles [25]. In contrast, our participant pool consisted of software engineering and computer science students. In our participants, we observed a greater inclination for tinkering, high self-efficacy and more comprehensive information processing. In samples with skewed distributions, using the median may inaccurately allocate a person to another end of the spectrum simply because they are, for example, more Abi-like than other participants in the sample. Misalignments could have also stemmed from the interplay of multiple facets. These findings underscore the importance of examining diverse populations and exploring how different facets interact, highlighting the need for future work to refine and extend GenderMag.

6.3. Implications

Implications for Tools: Tools are essential and crucial for software engineering and if they pose barriers for people with certain cognitive styles it becomes unjust and inequitable as some people will face a "cognitive tax" while using the tool. IDEs like Visual Studio Code, IntelliJ IDEA, and Eclipse combine essential functionalities, such as code editing, debugging and version controlling, into an integrated platform. Debugging, in particular, is an essential practice that supports developers in reading and understanding code [26]. Our results highlight that debugging tools, due to issues of discoverability and learnability, may fail to accommodate diverse cognitive styles. Consequently, they can disadvantage certain users beyond the typical challenges faced by newcomers. The inclusivity bugs we identified provides actionable insights on how debuggers can be made more inclusive. For example, if a debugger is started without any breakpoints, the user could be notified. This would benefit Abi-like process-oriented learners to know that the debugger requires a breakpoint to start, and it would benefit Tim-like tinkerers to not get distracted trying to figure out why the debugger did not start. A debugging study with students reported that debugging is usually not the first choice to find bugs. This is most likely due to general lack of familiarity with debuggers, compounded with challenges associated with using them [49]. They also found that tinkering can be advantageous, as it encourages curiosity and exploration of the debugger; however, excessive tinkering can become unproductive and waste time. Another recent study using a commonly used debugger in education, the PythonTutor debugger, found that participants who slowly stepped through every line of code performed better than those who were more selective and skipped lines [23]. One participant in their study mentions, *"I feel like, with the whole like visualization, it gets a little confusing for me."* This again underscores the importance of accommodating diverse cognitive styles. Beyond implications for performance, these findings also point to issues of cognitive inclusiveness that may stem from the design of the software tool itself.

In our future work, we plan to design fixes for some of the inclusivity bugs we have identified and to perform further experiments to validate whether those fixes have improved the inclusivity of the debugger.

Implications for Researchers: There is a still a lack of understanding on the cognitive inclusivity in many different tools used in SE. Many of the tools have been developed and improved over the years without empirical evidence. Prior research has examined inclusivity bugs through the lens of the GenderMag personas. We have also shown that lab experiments, using a think-aloud protocol, can be successful in identifying inclusivity bugs. Given the large number of inclusivity bugs found in our study and prior studies, we amplify the call to action made by Mendez et al. [11] and encourage more research to investigate how further software engineering tools can be made more cognitively inclusive.

Implications for Educators: Debugging in particular is given very little to no emphasis in courses [29,30,50]. Our findings reveal difficulties faced by newcomers when debugging. These insights can be used to improve curriculum around debugging. Moreover, educators must be aware of cognitive differences among students. Tailoring teaching methods to support these differences, and making students aware of the existing inclusivity bugs in debuggers can improve outcomes for students. Some of our participants expressed a lack of confidence and were quick to give up or blame themselves when they faced with challenges with the debugger. By helping students to understand that the tool itself is not supporting all ways of thinking can foster a more supportive learning environment.

Implications for Practitioners: Software practitioners should also be aware of the differences in cognitive styles and how newcomer barriers exist in many SE tools. This can result in an inclusive onboarding process. This is not only a matter of fairness but also one of fostering a diverse and thriving workforce. It also raises awareness about how software tools are used and draws attention to different cognitive processing styles that have not been known before.

6.4. Threats to validity

In this section, we discuss the threats to validity of the study.

Construct validity: Our experiment was conducted in a controlled lab setting with an observer, which helped ensure consistency across participants. However, this controlled environment may not fully reflect how developers debug in real-world settings. Factors such as having an observer present and knowing that the tasks had a set completion point may have influenced how participants approached the tasks. This limits the construct validity of our findings, as debugging behaviors might differ in more naturalistic settings. We encourage future field research of inclusivity of debuggers.

While we did pilot the study, we had only four participants in the pilot, which was not sufficient to cover all variations of cognitive styles of potential participants. Tasks 1 and Task 2 served as a warm-up to familiarize participants with the IDE and debugger. We acknowledge that warm-up tasks may affect participants differently depending on their cognitive styles. It is possible that these warm-up tasks may have reduced the number of inclusivity bugs for some cognitive style types, which caused us to underestimate the number of inclusivity bugs that would be faced by some newcomers in real world scenarios. Replication studies can further validate our results across more diverse participants.

Also a potential construct validity threat arises from the focus of this study on cognitive differences based on gender differences. GenderMag is designed to identify cognitive factors that influence problem-solving, but it particularly investigating gender differences. We acknowledge the need for additional studies to examine additional diversity dimensions like socioeconomic status and neurodivergence to enhance construct validity.

As mentioned in Section 6.2, we addressed the limitation of the median-based tagging of personas using quartiles. But this can cause a threat to validity, as this approach may still not accurately capture the personas.

Another threat to construct validity, as common in think-aloud studies, is that participants' verbalizations may not have fully captured their internal reasoning processes, and speaking aloud may have influenced their natural debugging behavior. Some participants went completely silent and had to be prompted to think out loud. Also, the interpretation of think-aloud data introduces a degree of subjectivity, despite our efforts to apply consistent coding practices.

Internal validity: The experiment was conducted in a controlled lab setting. Participants had no to minimal debugging experience. All participants were Software Engineering or Computer Science students, so they had some prior knowledge of debugging concepts. However, participants had varying levels of programming experience. To mitigate this, the tasks were designed with simple syntaxes to keep the focus on debugging rather than programming skills. However, we cannot exclude the presence of confounding factors.

External validity: The relatively small number of participants (24 in total), necessary for the tractability of qualitative analysis, poses a limitation to generalizability. In addition, all participants were self-selected and were students at the University of Auckland. The sample may not be representative of all newcomers. To mitigate this, we conducted the experiment in phases and stopped collecting data only after reaching theoretical saturation with no new inclusivity bugs being identified in a data collection round. Despite this, we do not know if the findings generalize beyond our participant sample as discussed in Section 6.2.

The debugging tasks were designed to reflect realistic issues, but may not fully capture the complexity and diversity of real-world debugging scenarios. This is an inherent threat in all lab experiments. Again, we encourage future field studies to validate our findings. Also, the experiment was conducted using a specific tool, the debugger integrated into the PyCharm IDE. While most of the features are common across all debugging tools, we do not know how our results generalize to other debuggers.

Although our sample included diverse cognitive styles, future research should examine whether these findings generalize to broader populations, including individuals with diverse gender identities, neurodiverse participants, etc. We emphasize that the insights presented here are not prescriptive design rules but rather empirically grounded directions that can inform further work. Notably, past efforts to redesign tools based on similar studies have successfully improved inclusivity across cognitive styles [3,5,6], underscoring the potential of our findings to contribute in similar ways. We therefore encourage future research to replicate and extend this study while being mindful of the contextual factors that shape cognitive inclusivity.

Conclusion validity: Given the small sample size, we were unable to examine statistical differences across the GenderMag facets. Our qualitative analysis provided a richer understanding of the inclusivity bugs. Future research could employ other quantitative methods. For example, now that we have identified a set of inclusivity bugs, an online questionnaire could be designed to probe the impact of these bugs across a larger population, enabling statistical analysis across GenderMag facets.

7. Conclusion

In this study, we observed 24 students with little to no experience in debugging perform some debugging tasks in a controlled lab setting. We found 21 inclusivity bugs in the PyCharm debugger. The inclusivity bugs had the two causes — discoverability and learnability. We showed that these inclusivity bugs affect certain cognitive characteristics, specifically those aligning towards the GenderMag Abi and Pat personas. However, the Tim persona was disadvantaged for one of the GenderMag facets, information processing style. Our insights shed light on the lack of inclusivity in SE tools and their impact on the fairness across diverse cognitive styles. As we move forward, our future work will focus on addressing these issues and evaluating the effectiveness of fixing these inclusivity bugs to improve fairness in SE tools.

CRediT authorship contribution statement

Faith Culas: Writing – original draft, Visualization, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Amisha Singh:** Methodology, Data curation, Conceptualization. **Atharva Arankalle:** Methodology, Data curation, Conceptualization. **Priyanka Dhopade:** Writing – review & editing, Supervision. **Kelly Blincoe:** Writing – review & editing, Supervision, Funding acquisition.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Faith Culas reports financial support was provided by Royal Society of New Zealand. Kelly Blincoe reports financial support was provided by Royal Society of New Zealand. Kelly Blincoe is an elsevier editorial member If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by a Rutherford Discovery Fellowship, administered by the Royal Society Te Apārangi, New Zealand.

Data availability

The data that has been used is confidential.

References

[1] C. Sadowski, E. Söderberg, L. Church, M. Sipko, A. Bacchelli, Modern code review: A case study at google, in: 2018 IEEE/ACM 40th International Conference on Software Engineering - Software Engineering in Practice Track, Icse-Seip 2018, 2018, pp. 181–190, http://dx.doi.org/10.1145/3183519.3183525.

[2] D. Ford, J. Smith, P.J. Guo, C. Parnin, Paradise unplugged: Identifying barriers for female participation on stack overflow, in: Fse'16: Proceedings of the 2016 24th Acm Sigsoft International Symposium on Foundations of Software Engineering, 2016, pp. 846–857, http://dx.doi.org/10.1145/2950290.2950331.

[3] I. Santos, J.F. Pimentel, I. Wiese, I. Steinmacher, A. Sarma, M.A. Gerosa, Designing for cognitive diversity: Improving the GitHub experience for newcomers, in: 2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Society, ICSE-SEIS, IEEE, 2023, pp. 1–12, http://dx.doi.org/10.1109/icse-seis58686.2023.00007.

[4] C. Mendez, H.S. Padala, Z. Steine-Hanson, C. Hilderbrand, A. Horvath, C. Hill, L. Simpson, N. Patil, A. Sarma, M. Burnett, Open source barriers to entry, revisited: a sociotechnical perspective, in: Proceedings of the 40th International Conference on Software Engineering, ICSE '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 1004–1015, http://dx.doi.org/10.1145/3180155.3180241.

[5] M. Guizani, I. Steinmacher, J. Emard, A. Fallatah, M. Burnett, A. Sarma, How to debug inclusivity bugs? in: Proceedings of the 2022 ACM/IEEE 44th International Conference on Software Engineering: Software Engineering in Society, ACM, 2022, pp. 90–101, http://dx.doi.org/10.1145/3510458.3513009.

[6] E. Murphy-Hill, A. Elizondo, A. Murillo, M. Harbach, B. Vasilescu, D. Carlson, F. Dessloch, GenderMag improves discoverability in the field, especially for women, in: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, ACM, 2024, http://dx.doi.org/10.1145/3597503.3639097.

[7] M.H. Ashcraft, Cognition, Prentice Hall, 2002.

[8] F. Fagerholm, M. Felderer, D. Fucci, M. Unterkalmsteiner, B. Marculescu, M. Martini, L.G.W. Tengberg, R. Feldt, B. Lehtelä, B. Nagyváradi, J. Khattak, Cognition in software engineering: A taxonomy and survey of a half-century of research, ACM Comput. Surv. 54 (11s) (2022) 1–36, http://dx.doi.org/10.1145/3508359.

[9] M. Burnett, S.D. Fleming, S. Iqbal, G. Venolia, V. Rajaram, U. Farooq, V. Grigoreanu, M. Czerwinski, Gender differences and programming environments, in: Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ACM, 2010, http://dx.doi.org/10.1145/1852786.1852824.

[10] M. Vorvoreanu, L.Y. Zhang, Y.H. Huang, C. Hilderbrand, Z. Steine-Hanson, M. Burnett, From gender biases to gender-inclusive design: An empirical investigation, in: Chi 2019: Proceedings of the 2019 Chi Conference on Human Factors in Computing Systems, 2019, http://dx.doi.org/10.1145/3290605.3300283.

[11] C. Mendez, A. Sarma, M. Burnett, Gender in open source software: What the tools tell, in: Proceedings of the 1st International Workshop on Gender Equality in Software Engineering, ACM, 2018, pp. 21–24, http://dx.doi.org/10.1145/3195570.3195572.

[12] A. Zeller, Why Programs Fail: A Guide to Systematic Debugging, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[13] S. Fitzgerald, G. Lewandowski, R. McCauley, L. Murphy, B. Simon, L. Thomas, C. Zander, Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers, Comput. Sci. Educ. 18 (2) (2008) 93–116, http://dx.doi.org/10.1080/08993400802114508.

[14] H.A. Witkin, C.A. Moore, D.R. Goodenough, P.W. Cox, Field-dependent and field-independent cognitive styles and their educational implications, Rev. Educ. Res. 47 (1) (1977) 1–64, URL http://www.jstor.org/stable/1169967.

[15] J. Kagan, Reflection-impulsivity and reading ability in primary grade children, Child Dev. 36 (3) (1965) 609–628, URL http://www.jstor.org/stable/1126908.

[16] J.S. Hyde, The gender similarities hypothesis, Am. Psychol. 60 (6) (2005) 581–592, http://dx.doi.org/10.1037/0003-066x.60.6.581, Hyde, Janet Shibley Comparative Study Journal Article Research Support, U.S. Gov't, Non-P.H.S. Review United States 2005/09/22 Am Psychol. 2005 Sep;60(6):581-592. doi: 10.1037/0003-066X.60.6.581..

[17] C.M. Steele, A threat in the air. How stereotypes shape intellectual identity and performance, Am. Psychol. 52 (6) (1997) 613–629, http://dx.doi.org/10.1037//0003-066x.52.6.613, Steele, C M MH51977/MH/NIMH NIH HHS/United States Journal Article Research Support, Non-U.S. Gov't Research Support, U.S. Gov't, P.H.S. United States 1997/06/01 Am Psychol. 1997 Jun;52(6):613-29. doi: 10.1037//0003-066x.52.6.613..

[18] A. Saini, Inferior: How Science Got Women Wrong-and the New Research That's Rewriting the Story, Beacon Press, 2017, URL https://books.google.co.nz/books?id=U5DEDgAAQBAJ.

[19] M. Burnett, S. Stumpf, J. Macbeth, S. Makri, L. Beckwith, I. Kwan, A. Peters, W. Jernigan, GenderMag: A method for evaluating software's gender inclusiveness, Interact. Comput. 28 (6) (2016) 760–787, http://dx.doi.org/10.1093/iwc/iwv046.

[20] N. Marsden, M. Haag, Stereotypes and politics: Reflections on personas, in: Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, ACM, 2016, pp. 4017–4031, http://dx.doi.org/10.1145/2858036.2858151.

[21] C.G. Hill, M. Haag, A. Oleson, C. Mendez, N. Marsden, A. Sarma, M. Burnett, Gender-inclusiveness personas vs. Stereotyping: Can we have it both ways? in: Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, CHI '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 6658–6671, http://dx.doi.org/10.1145/3025453.3025609.

[22] L. O'Brien, T. Kanij, J. Grundy, Assessing gender bias in the software used in computer science and software engineering education, J. Syst. Softw. 219 (2025/01/01) http://dx.doi.org/10.1016/j.jss.2024.112225, URL https://www.sciencedirect.com/science/article/pii/S0164121224002693?via%3Dihub.

[23] M. Hassan, G. Zeng, C. Zilles, Evaluating how novices utilize debuggers and code execution to understand code, in: Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1, vol. 5, ACM, 2024, pp. 65–83, http://dx.doi.org/10.1145/3632620.3671126.

[24] R. Minelli, A. Mocci, M. Lanza, I know what you did last summer - an investigation of how developers spend their time, in: 2015 IEEE 23rd International Conference on Program Comprehension, 2015, pp. 25–35, http://dx.doi.org/10.1109/ICPC.2015.12.

[25] M. Burnett, A. Peters, C. Hill, N. Elarief, Finding gender-inclusiveness software issues with GenderMag: A field investigation, in: 34th Annual Chi Conference on Human Factors in Computing Systems, Chi 2016, 2016, pp. 2586–2598, http://dx.doi.org/10.1145/2858036.2858274.

[26] W. Maalej, R. Tiarks, T. Roehm, R. Koschke, On the comprehension of program comprehension, ACM Trans. Softw. Eng. Methodol. 23 (4) (2014) 1–37.

[27] T. Michaeli, R. Romeike, Improving debugging skills in the classroom, in: Proceedings of the 14th Workshop in Primary and Secondary Computing Education, ACM, http://dx.doi.org/10.1145/3361721.3361724.

[28] S.N. Liao, S. Valstar, K. Thai, C. Alvarado, D. Zingaro, W.G. Griswold, L. Porter, Behaviors of higher and lower performing students in CS1, in: Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 196–202, http://dx.doi.org/10.1145/3304221.3319740.

[29] Y. Noller, E. Chandra, S. HC, K. Choo, C. Jegourel, O. Kurniawan, C.M. Poskitt, Simulated interactive debugging, 2025, arXiv:2501.09694.

[30] A. Kanaya, T. Migo, H. Hashiura, A proposal for a debugging learning support environment for undergraduate students majoring in computer science, 2024, arXiv:2407.17743.

[31] A. Deiner, G. Fraser, NuzzleBug: Debugging block-based programs in scratch, in: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, ACM, pp. 1–13, http://dx.doi.org/10.1145/3597503.3623331.

[32] Y.-T. Lin, C.-C. Wu, T.-Y. Hou, Y.-C. Lin, F.-Y. Yang, C.-H. Chang, Tracking students' cognitive processes during program debugging—An eye-movement approach, IEEE Trans. Educ. 59 (3) (2016) 175–186, http://dx.doi.org/10.1109/te.2015.2487341.

[33] V. Grigoreanu, M. Burnett, S. Wiedenbeck, J. Cao, K. Rector, I. Kwan, End-user debugging strategies: A sensemaking perspective, ACM Trans. Comput.-Hum. Interact. 19 (1) (2012) http://dx.doi.org/10.1145/2147783.2147788.

[34] M.E. Fonteyn, B. Kuipers, S.J. Grobe, A description of think aloud method and protocol analysis, Qual. Health Res. 3 (4) (1993) 430–441, http://dx.doi.org/10.1177/104973239300300403.

[35] E. Tempero, P. Denny, J. Finnie-Ansley, A. Luxton-Reilly, D. Kirk, J. Leinonen, A. Shakil, R. Sheehan, J. Tizard, Y.-C. Tu, B. Wünsche, On the comprehensibility of functional decomposition: An empirical study, in: 2024 Ieee/Acm 32nd International Conference on Program Comprehension, ICPC 2024, 2024, http://dx.doi.org/10.1145/3643916.3644432.

[36] N. Alzahrani, F. Vahid, Common logic errors for programming learners: A three-decade literature survey, in: ASEE Virtual Annual Conference Content Access, ASEE Conferences, Virtual Conference, 2021, URL https://peer.asee.org/36814.

[37] F. Culas, A. Singh, A. Arankalle, P. Dhopade, K. Blincoe, Replication package for "newcomer experience during debugging: A cognitive inclusivity perspective", 2025, Not public yet., https://auckland.figshare.com/collections/Replication_package_-_Newcomer_Experience_during_Debugging/7756889.

[38] M.M. Hamid, A. Chatterjee, M. Guizani, A. Anderson, F. Moussaoui, S. Yang, I. Escobar, A. Sarma, M. Burnett, How to measure diversity actionably in technology, in: D. Damian, K. Blincoe, D. Ford, A. Serebrenik, Z. Masood (Eds.), Equity, Diversity, and Inclusion in Software Engineering: Best Practices and Insights, A Press, Berkeley, CA, 2024, pp. 469–485, http://dx.doi.org/10.1007/978-1-4842-9651-6_27.

[39] G. Guest, A. Bunce, L. Johnson, How many interviews are enough?: An experiment with data saturation and variability, Field Methods 18 (1) (2006) 59–82, http://dx.doi.org/10.1177/1525822X05279903.

[40] V. Braun, V.C. and, Using thematic analysis in psychology, Qual. Res. Psychol. 3 (2) (2006) 77–101, http://dx.doi.org/10.1191/1478088706qp063oa, arXiv:https://www.tandfonline.com/doi/pdf/10.1191/1478088706qp063oa, URL https://www.tandfonline.com/doi/abs/10.1191/1478088706qp063oa.

[41] V. Braun, V.C. and, One size fits all? What counts as quality practice in (reflexive) thematic analysis? Qual. Res. Psychol. 18 (3) (2021) 328–352, http://dx.doi.org/10.1080/14780887.2020.1769238.

[42] P. Pirolli, S. Card, The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis, 2005.

[43] M. Burnett, S. Stumpf, J. Macbeth, S. Makri, L. Beckwith, I. Kwan, A. Peters, W. Jernigan, GenderMag: A method for evaluating software2019s gender inclusiveness, Interact. Comput. 28 (6) (2016) 760–787, http://dx.doi.org/10.1093/iwc/iwv046.

[44] C. Mendez, L. Letaw, M. Burnett, S. Stumpf, A. Sarma, C. Hilderbrand, From GenderMag to InclusiveMag: An inclusive design meta-method, 2019, arXiv:1905.02812.

[45] L. Witzel, Extending and Applying GenderMag with a Transgender and a Non-Binary Persona (Bachelor's thesis), University of Zurich, 2020, URL: https://capuana.ifi.uzh.ch/publications/PDFs/20600_Bachelorarbeit.pdf.

[46] C. Richards, W.P. Bouman, L. Seal, M.J. Barker, T.O. Nieder, G. T'Sjoen, Non-binary or genderqueer genders, Int. Rev. Psychiatry 28 (1) (2016) 95–102, http://dx.doi.org/10.3109/09540261.2015.1106446.

[47] D. Joel, Z. Berman, I. Tavor, N. Wexler, O. Gaber, Y. Stein, N. Shefi, J. Pool, S. Urchs, D.S. Margulies, F. Liem, J. Hänggi, L. Jäncke, Y. Assaf, Sex beyond the genitalia: The human brain mosaic, Proc. Natl. Acad. Sci. 112 (50) (2015) 15468–15473, http://dx.doi.org/10.1073/pnas.1509654112, arXiv:https://www.pnas.org/doi/pdf/10.1073/pnas.1509654112, URL https://www.pnas.org/doi/abs/10.1073/pnas.1509654112.

[48] T. Morgenroth, M.K. Ryan, The effects of gender trouble: An integrative theoretical framework of the perpetuation and disruption of the gender/sex binary, Perspect. Psychol. Sci. 16 (6) (2021) 1113–1142, http://dx.doi.org/10.1177/1745691620902442, PMID: 32375012.

[49] S. Fitzgerald, R. Mccauley, B. Hanks, L. Murphy, B. Simon, C. Zander, Debugging from the student perspective, IEEE Trans. Educ. 53 (3) (2010) 390–396, http://dx.doi.org/10.1109/te.2009.2025266.

[50] J. Whalley, A. Settle, A. Luxton-Reilly, A think-aloud study of novice debugging, ACM Trans. Comput. Educ. 23 (2) (2023) 1–38, http://dx.doi.org/10.1145/3589004.