AID: An automated detector for gender-inclusivity bugs in OSS project pages

Amreeta Chatterjee[1], Mariam Guizani[1], Catherine Stevens[1], Jillian Emard[1], Mary Evelyn May[1], Margaret Burnett[1], Iftekhar Ahmed[2], Anita Sarma[1]
[1]*Oregon State University, Corvallis, OR 97330, USA*, [2]*University of California, Irvine, Irvine, CA*

{chattera,guizanim,stevecat,emardj,mayma}@oregonstate.edu, burnett@eecs.oregonstate.edu, iftekha@uci.edu, anita.sarma@oregonstate.edu

2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE) | 978-1-6654-0296-5/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICSE43902.2021.00128

***Abstract*—The tools and infrastructure used in tech, including Open Source Software (OSS), can embed "inclusivity bugs"—features that disproportionately disadvantage particular groups of contributors. To see whether OSS developers have existing practices to ward off such bugs, we surveyed 266 OSS developers. Our results show that a majority (77%) of developers do not use any inclusivity practices, and 92% of respondents cited a lack of concrete resources to enable them to do so. To help fill this gap, this paper introduces AID, a tool that automates the GenderMag method to systematically find gender-inclusivity bugs in software. We then present the results of the tool's evaluation on 20 GitHub projects. The tool achieved precision of 0.69, recall of 0.92, an F-measure of 0.79 and even captured some inclusivity bugs that human GenderMag teams missed.**

***Index Terms*—Gender inclusivity, automation, open source, information processing**

I. Introduction

A new class of bug is beginning to gain visibility in software engineering literature: the inclusivity bug. An inclusivity bug is software behavior that disproportionately disadvantages a particular group of users of that software. To date, the most studied type of inclusivity bugs are gender-inclusivity bugs, most (but not all) of which disproportionately disadvantage women [1]–[8].

Some gender-inclusivity bugs in software manifest themselves overtly, such as with non-inclusive language [9], [10], gender-stereotyped game characters [11], or gender-stereotyped job matches [12], [13]. Detecting overt gender-inclusivity bugs is relatively straightforward, by definition of the term "overt" (although actually fixing them is not always straightforward). However, non-overt gender-inclusivity bugs are not so obvious, such as software behaviors and workflows that are biased against cognitive or behavioral styles more common in one gender than another. For example, among the software-relevant gender differences pointed out in Stumpf et al.'s survey are gender clusters of differences in the colors and sounds people can distinguish, how people comprehend and interpret verbal/written communications, their spatial processing, their attitudes toward technological risks, and how they process information [10].

The extent to which such non-overt gender-inclusivity bugs permeate software is large. The *lowest* percentage we have been able to locate is one team's report of inclusivity bugs in "only" 12% of the features they considered [14]; other teams have reported much higher rates of inclusivity bugs [4], [7], [14]. For example, in one recent study on a group of OSS projects, the rate of gender-inclusivity bugs reported by OSS professionals that were then verified by OSS newcomers was 63% [15].

Fortunately, there are methods that can help software professionals find, fix, and/or avert inclusivity bugs (e.g., [16], [17]). One of these is GenderMag (*Gender* Inclusiveness *Mag*nifier) [3], which provides the foundation of the tool presented in this paper. GenderMag is a method that enables developers to systematically find gender-inclusivity bugs in their software or workflow so that they can fix the bugs. GenderMag variants have been used in a variety of domains, including digital libraries [18], learning tools and websites [3], [14], [19], [20], machine learning interfaces [14], robotics [6], search engines [7], and OSS projects [15], [21].

Unfortunately, however, using any of these methods can be expensive, because they are entirely manual and labor-intensive. For example, GenderMag's creators recommend that at least 3 evaluators spend 1-2 hours per session [22], where a session usually covers only 1-3 use-cases. In fact, finding ways to reduce the cost was a major theme in a recent field study of 10 teams' work to integrate GenderMag into their development processes [20].

Can a software tool help to address this problem, to increase the viability of debugging inclusivity bugs that lurk in software? To find out, we conducted a survey of experienced OSS developers (OSS'ers), to learn what tools and other inclusivity debugging resources developers actually use.

Informed by the study, we then created AID, a tool that automates a "vertical slice" of GenderMag. We used a vertical slice instead of all of GenderMag for two reasons. First, since this is the first of a new class of software tools (inclusivity bug detectors), we needed a tractable way to investigate whether this class of tools is even possible. Second, it enabled us to empirically investigate AID's effectiveness under the strict controls that a single vertical slice affords. Specifically, this version of the tool uses a portion of *all* the components of GenderMag—a portion of its scope (OSS GitHub projects), one of its personas (Abi), and one of its cognitive styles (the information processing style). We then empirically investigated

978-1-6654-0296-5/21/$31.00 ©2021 IEEE
DOI 10.1109/ICSE43902.2021.00128

TABLE I
THE FACETS & ABI'S VALUES FOR EACH (SEE [3] FOR THE FOUNDATIONS BEHIND EACH)

Facet	Abi's Values
Motivations	Uses technology to accomplish tasks
Computer Self-Efficacy	Low compared to peers
Attitude towards Risk	Risk-averse
Information Processing Style	Comprehensive
Learning Style	Learns by process

the tool's results on 20 OSS GitHub projects.

This paper presents AID and reports our empirical findings. The contributions of this paper are:

- A survey of 266 experienced OSS developers' inclusivity debugging practices, and the tools/guidelines they use for these practices;
- AID, the first software tool to automate the detection of inclusivity bugs; and
- An empirical investigation of AID's efficacy on 20 OSS GitHub projects.

II. BACKGROUND: THE GENDERMAG METHOD

AID is a partial automation of the GenderMag method [3]. GenderMag is a process that enables software professionals to systematically locate gender-inclusivity "bugs" in the user-facing portions of their software. Locating these bugs is a prerequisite to fixing them. GenderMag has been used successfully in a variety of domains, including university webware, educational software, machine-learning interfaces, mobile apps, digital libraries, search engines, and software tools [3], [7], [14], [18]–[21], [23].

At the core of GenderMag method are five problem solving styles, or *facets*, each backed by extensive foundational research [3], [10]. The five facets are: Motivations for using technology, Computer Self-Efficacy, Attitude towards Risk, Information Processing Style, and Learning Style. Each facet has a range of possible values, and GenderMag uses three multi-personas (Abi, Pat, and Tim) to bring different facet values to life (Table I and Figure 1). Each persona embodies a different set of values for the five facets. Facet values statistically more common among women are assigned to Abi, those more common among men are assigned to Tim, and a mix of various facet values are assigned to Pat. These facet values are what define each persona's problem-solving style. Other persona attributes (e.g., job, age, hobbies, preferred pronouns, etc...) are customizable.

Of particular interest to this paper is the Information Processing facet as it pertains to the Abi persona, because the tool we present is a partial automation of that facet from Abi's perspective. The information processing facet describes how a user receives new information to make decisions about actions to take, in order to accomplish a task. The Abi persona processes information comprehensively—that is, by gathering fairly complete information before proceeding. At the opposite end of the information processing spectrum of values is the Tim persona, who is more likely to process information selectively and act upon the first promising information they find, then backtrack if needed [24].

To conduct a GenderMag session, a software team "channels" one of these personas, chooses a use-case (a task for the persona to try to accomplish), and walks through that use-case (task) using a specialized version of the Cognitive Walkthrough (CW) family [25]. In this process, the team answers one question for each subgoal and two questions for each action that the software team is hoping a user will take to accomplish the task. The "action" questions are answered before and after performing the action. The questions are:

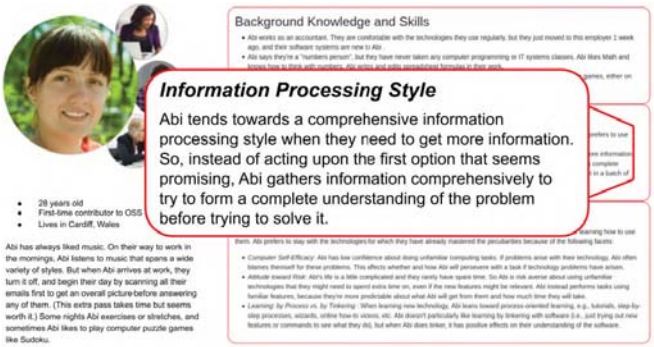

Fig. 1. The Abi persona, with their information processing style facet value enlarged for readability. (See supplemental materials at [26] for the full version.)

- Subgoal Question (S-Q): Will <persona> have formed this subgoal as a step to their overall goal? (Yes/No/Maybe, why, what facets did you use)
- Action Question#1 (A-Q#1): Will <persona> know to take the action? (Yes/No/Maybe, why, what facets did you use)
- Action Question#2 (A-Q#2): If <persona> takes the action, will they know they did the right thing and are making progress toward their goal? (Yes/No/Maybe, why, what facets did you use)

If the team answers as No/Maybe to any of these questions and attributes it to one of the <persona>'s facet, then (and only then) it is considered an inclusivity bug. AID automates the process described above for Abi's information processing facet value.

III. FORMATIVE STUDY

To understand (1) the state of inclusivity evaluations in OSS and (2) the challenges behind incorporating such evaluations in the development process, we conducted a formative investigation in the form of a survey.

A. Survey Methodology

We recruited OSS developers as survey participants as our goals were to learn about inclusivity evaluation practices and needs inside OSS. We aimed to get the perspective of experienced OSS developers since in this version of AID we are using GitHub-based project pages as our test subjects.

TABLE II
INCLUSIVITY PRACTICE SURVEY DATA GROUPED BY DEMOGRAPHICS.

		Yes 23% (61 of 266)	No 77% (205 of 266)
Size	1-49 (102 respondents)	18%	82%
	50-999 (65 respondents)	25%	75%
	1,000-5,000 (34 respondents)	21%	79%
	>5,000 (65 respondents)	31%	69%
Experience	<3 months (7 respondents)	43%	57%
	3-6 months (5 respondents)	0%	100%
	7 mo.-1 yr. (4 respondents)	0%	100%
	>1 year (250 respondents)	23%	77%
Participation Type	Paid (221 respondents)	24%	76%
	Unpaid (45 respondents)	20%	80%

When recruiting participants we used the number of followers as a filtering criterion since experienced OSS developers are more likely than newcomers to accrue followers. For example, Blincoe et al. found that popular GitHub OSS'ers influence their followers and attract them to new projects [27].

We used the number of followers as a filter criterion because experienced OSS developers are more likely than newcomers to accrue followers. For example, Blincoe et al. found that popular GitHub OSS'ers influence their followers and attract them to new projects [27].

In order to do that, we used the GitHub search tool to retrieve 5,000 usernames that are among the top-followed GitHub OSS'ers. To get the contacts of this list 5,000 usernames, we modified a tool called Zen [28], which obtained email addresses for 3,124 of these usernames. We sent the survey to these 3,124 experienced OSS developers.

The survey had a total of eight questions, comprising Yes/No, multiple choice, and open-ended questions. In addition to collecting demographic information, the survey used branching logic when asking about existing inclusivity evaluation practices. If the participant indicated that their development practices include doing some form of inclusivity evaluation, we asked more about these; otherwise we asked what challenges prevent them from doing so. In pilot runs, the survey took about five minutes to complete.

We emailed this survey to 3,124 developers. 253 of the emails bounced or were unreachable, which resulted in 2,871 recipients. At the end of the two weeks during which the survey remained open, we received 394 responses, or a response rate of 13.7% (394/2,871). This response rate compares favorably to the 7.9% survey response rate found in a previous software engineering study [29]. We removed the survey responses that had partial answers, leaving us with 266 responses. The survey questions and the tool used to retrieve the developers' email addresses are provided at [26].

B. Survey Results

Over three-fourths (77%) of the survey respondents reported that they do not incorporate inclusivity practices into their software development processes. As Table II's rightmost column shows, the majority of every demographic group reported not using inclusivity practices.

When those who responded "No" were asked why, they responded as summarized in Figure 2. Almost half the reasons related to lack of awareness/interest. Specifically, about 30% of the reasons were a lack of awareness of such practices or of the importance of doing so, another 12% reported lack of support from management, and 13% ("other, please specify") explained that such practices were up to individual developers, or that they were unnecessary, unimportant, or even harmful:

P84: *"Inclusivity happens on an ad-hoc basis and depends on the individual engineer..."*

P123: *"Everyone I work with uses internet handles. I have no clue if 'LadyOTheLake', is a guy, gal, minority, straight, gay, ... We just code and make good stuff."*

P145: *"Open source communities have never prioritized inclusivity."*

The above reasons are perhaps unsurprising, given that the concept of finding/fixing inclusivity bugs within technology itself, beyond accessibility (e.g., providing alt text), is a relatively recent one in software engineering. For example, to the best of our knowledge, the first paper in the software engineering community to even mention the *concept* of software's inclusivity bugs was only a decade ago (2010) [30], and the first software engineering presentation of a systematic *process* for software developers to find such bugs wasn't until 2016 [31].

Reasons like the above are, at best, only indirectly addressable by creating a new tool. However, the remaining 46% of the reasons survey participants gave—lack of specific guidelines (26%), the costs of inclusivity evaluation (12%), and lack of tools (8%)—are all potential fodder for software engineering tools.

In fact, even the 23% of respondents who did report using inclusivity practices in their development processes did not have much in the way of concrete techniques in their existing inclusivity efforts. Only 28 of these Yes-respondents (less than one-third of the Yes's) reported using a technique of any sort.

Of these 28 Yes-respondents, nine use mainly ad-hoc inclusivity techniques (Figure 2, bottom four bars), such as inspecting their work for stereotyping language or character depictions; relying on user feedback or user study results; or following general codes of conduct. Another nine relied on internal/custom tools (Figure 2, second bar from the top).

Only four public inclusivity techniques were reported by *any* respondents: GenderMag with/without the GenderMag Recorder's Assistant (10 respondents), public guidelines for inclusive language in code [32] (one respondent), public guidelines for inclusive documentation [33] (one respondent), and public guidelines for ethical collection of training data [34] (one respondent). GenderMag was the only inclusivity technique used by multiple respondents.

The above 28 respondents—the only respondents who reported using any specific techniques, tools, or guidelines—are only 8% of the total respondents. The other Yes's instead described a lack of tools and/or guidelines in their inclusivity efforts:

P299: *"We don't use specific techniques..."*

P130: *"Lack of knowledge and tools makes developers reinventing the wheels all the time."*

P60: *"...there are no such guidelines or tools ... I hope that just as we have guidelines for code quality we [could] also have inclusivity principles..."*

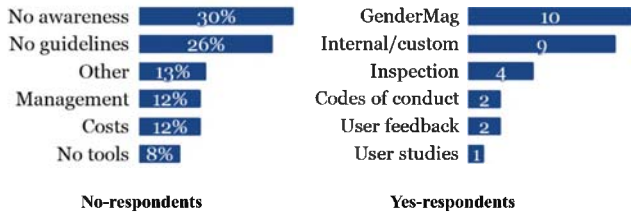

Fig. 2. (Left): No-respondents' reasons/challenges for not using inclusivity, as a percentage of all challenges. (Note: Data shown totals 101%, due to rounding of each individual bar.) (Right): Yes-respondents' uses of different inclusivity techniques.

IV. AID

Even the few survey respondents who reported using a concrete technique had only a little ammunition to address three of the tool-relevant challenges alluded to above: costs, lack of tools, and lack of concrete guidelines. Regarding costs, GenderMag was the only publicly available method used, and it is costly, requiring teams of multiple software professionals to work together in sessions that can last two or more hours [20]. Regarding tools, other than a few mentions of internal tools, no respondent reported tool usage other than the Recorder's Assistant, which is simply a note-taking organizational aid. Regarding guidelines, a few respondents pointed to concrete guidelines, but none went beyond language use and stereotyping issues. Together, these three categories accounted for 46% of the challenges reported by survey participants who do not use any form of inclusivity evaluation (Figure 2), and were also pointed out by some participants who *do* use some kind of inclusivity evaluation.

To overcome these challenges, we created AID, a tool whose ultimate goal is to automatically detect non-overt gender inclusivity bugs like those described in Section I. In this paper we report on our first step—automating a vertical slice of GenderMag inclusivity evaluations—toward this ultimate goal.

UC	Subgoal	S-Q	Why	UI	Action	A-Q#1	Why	A-Q#2	Why
File a new issue	Find info about...	Yes	-	UI8	Read Readme.md	Yes	-	No	...[no] info...abou issues or filing n
				UI11	Click on "community	No	...no mention	No	... thinks that it t back to the sam
				UI4	Click on "How to file	Yes	-	No	...does not really mention how to f
	File the issue	No	...[not] enough info...	UI5	Click on "open issue"	Yes	-	Yes	-
				UI13	Click on "new issue"	Yes	-	Yes	-
				UI14	Click	Yes	-	Yes	-

Fig. 3. A snapshot of a spreadsheet with inputs and outputs from a manual GenderMag session (information processing facet only). AID uses the Subgoal and Action columns as input and produces similar outputs as the manual GenderMag session. Orange backgrounds: Use-case and Subgoal inputs; white backgrounds: Action inputs; blue backgrounds: GenderMag evaluation outputs.

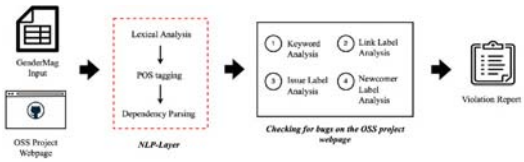

Fig. 4. Overview of AID's architecture

One choice that defined our vertical slice was the type of software we scoped our tool to. We scoped our tool to use *OSS platforms*, which is an important genre for inclusivity bug detection because (1) these platforms are the sole mechanism through which OSS community members can interact, and (2) OSS tools and technology have pervasive gender-inclusivity bugs [15], [21]. Another choice in defining our vertical slice was the persona. We chose GenderMag's *Abi* persona as per the "Abi first" practice, commonly used because Abi has historically provided the most powerful lens for revealing inclusivity bugs [20]. The third choice was the facet. We decided on the *information processing* style facet, because OSS projects are information-rich environments.

For this vertical slice, AID takes as inputs the use-cases, subgoals, and actions for a GenderMag session as well as the URLs of the project's GitHub pages (Figure 3). Using these inputs, AID evaluates the project's GitHub pages against its model and produces an inclusivity bug report similar to the blue region of Figure 3.

A. Deriving the model

The question of how to automate GenderMag presents a number of challenges. First, there are an infinite number of platforms and UIs that could be evaluated, each of which have an infinite number of use-cases. Second, as a member of the CW family, GenderMag involves simulating (other) humans, a task that some people find difficult to accomplish [35]. The few researchers who have automated CWs for limited purposes (e.g., [36]) have done so under the assumption that users are homogeneous, and have not attempted to take into account the range in people's cognitive styles. Finally, extensive GenderMag session data does not yet exist, which prevents some kinds of approaches from being viable. For example, a machine-learning approach would require training a model using data from millions of GenderMag sessions, including the context associated with each (platform, UI elements, etc.)

Since AID is the first investigation into whether an inclusivity bug detector is even possible, the tool's feature-completeness was not a critical consideration. Thus, we counteracted the first and second challenges via the vertical slice already discussed. To address the third challenge, we turned to a decision-rule approach.

An advantage of decision rules is that beyond statistical patterns, they can also harness relevant theories, such as those in GenderMag's foundational core (e.g., self-efficacy theory [37], information processing theory [24], etc.) Thus, they do not require nearly as much empirical data.

That said, we still needed *some* empirical data to ground our decision rules in OSS-relevant data—the inclusivity bugs in real OSS projects and why they occur (recall Figure 3). Toward that end, we obtained GenderMag session data from two active OSS teams (Projects F and J) that have already used GenderMag to improve their projects' inclusivity. Project F is a GitHub-based platform to help newcomers become familiar with OSS; it has 21 contributors and 2.5 million lines of code. Project J is a GitHub-based data science project with 295 contributors and 1.3 million lines of code. The project teams shared their GenderMag session materials, which included their use-cases, customized GenderMag personas, and evaluation outputs (recall Figure 3). Together these teams' data identified 19 inclusivity bugs related to Abi's information processing style.

To supplement the data obtained from the external projects, we sought out additional GitHub-based OSS projects to conduct additional GenderMag evaluations.

We turned to the classification by Fronchetti et al., which classified 450 OSS GitHub-based projects into three groups based on the average growth of newcomers: (1) Logarithmic growth (71 projects); (2) linear growth (322 projects); and (3) Exponential growth (57 projects) [38]. We randomly selected six projects from each group, resulting in 18 projects of varying maturity, size and domain (Table III).

We then manually conducted GenderMag sessions on them to produce data about the inclusivity bugs within these projects pertaining to the information processing style facet. In order to remain consistent with sessions from Projects F & J, we used the same Abi persona (which was customized to be a newcomer) and use-cases that they had used (Table IV). These sessions covered 255 actions and produced answers to a total of 510 questions—two questions per action (Section II). (The sessions also produced answers to 126 subgoal questions, but since none referred to information processing style, we did not include them in the analysis.) The answers to these questions revealed 257 inclusivity bugs—steps in the use-cases where OSS'ers with Abi's comprehensive information processing style would be disadvantaged. These 257 inclusivity bugs plus the 19 inclusivity bugs from the Projects F & J, a total of 276 inclusivity bugs, comprised the dataset from which we derived the model's decision rules.

B. Reliability Safeguards

We used multiple strategies to safeguard the reliability of the GenderMag session outputs (see Figure 3) used to infer our decision rules and the empirical results from AID.

1) Inter-Rater Reliability of GenderMag Session Outputs: The first safeguard was inter-rater reliability, to ensure consistency of the GenderMag session outputs we conducted on the 18 projects. Two pairs of researchers had each conducted GenderMag sessions. To measure consistency of their GenderMag session outputs, we drew upon inter-rater reliability calculations often used with qualitative analysis [39]. Specifically, each team independently ran GenderMag sessions on 20% of the use-cases. They then compared their outputs using the Jaccard index [40] to calculate a consensus, which resulted in 83% agreement. Given this level of team consensus [40], the two teams divided the remaining projects.

2) Validation against OSS Teams' GenderMag Session Outputs: The above is a measure of consistency between the two teams of researchers. To measure consistency with real OSS teams' GenderMag outputs, we validated our research teams' outputs against those from Projects F & J. Recall that Teams F & J conducted the GenderMag sessions themselves on their own projects.

We grouped the 19 inclusivity bugs from Projects F & J into 9 categories for cross validation purposes (Table VIII, which will be discussed further in Section V) through three rounds of negotiated agreement [41] among four researchers. Validating the GenderMag sessions we conducted against those of Projects F & J, 69% of the inclusivity bugs from the sessions we conducted matched the inclusivity bug categories of Projects F & J. Validating in the other direction, 79% of the inclusivity bugs from Projects F & J's GenderMag sessions matched the inclusivity bugs in the sessions we conducted.

3) Cross-validation of Tool Results: Finally, we cross-validated each individual inclusivity bug that AID identified for all 20 projects against the inclusivity bugs manually identified in GenderMag sessions on all 20 projects. The results of this cross-validation are in Section V.

C. Model: Decision rules

To create the decision rules driving AID, the first and second authors analyzed the GenderMag data from the 20 projects (F & J + 18 GenderMag'ed by the research team). They analyzed each inclusivity bug and the "whys" behind it, as well as the UI. Through inductive reasoning they abstracted the "whys" into a set of 11 decision rules. Then through three rounds of negotiated agreement they refined and merged similar rules to create the final set of five distinct rules, which form the foundation of AID.

AID attempts to emulate a "real" GenderMag session by taking as input a spreadsheet that lists the data that humans would use when performing a GenderMag session. The first three columns are the use-case, subgoal, and action inputs, and the fourth column is the webpage URL for that step in the CW. (Section II details a GenderMag session process.) Figure 4 displays the overview of AID's architecture and Figure 3 shows an example of an input sent to AID for the use-case "File an issue".

From this input, AID extracts the sentence structure from the text in the subgoal and action. It does so through a combination of Natural Language Processing (NLP) techniques, such as lexical analysis [42], part-of-speech tagging [43] and dependency parsing [44]. Overall, AID uses Python 3.7.4 and associated libraries (spaCy, BeautifulSoup, gensim, xlrd) to implement the decision rules. The code is available at [26].

The implementation details of each rule are discussed next.

1) Cues to needed information: The first rule in our set checks for situations where OSS'ers like Abi would not find all the information they need to complete their task. For

TABLE III
CHARACTERISTICS OF THE 18 PROJECTS THAT RESEARCHERS EVALUATED USING GENDERMAG

	Name	Programming Language	LOC	# Contributors	Age	Domain
Logarithmic Growth	imathis/octopress	Ruby	3,734	118	11	application software
	chaplinjs/chaplin	CoffeeScript	13,390	82	8	web libraries and frameworks
	antirez/disque	C	68,192	54	5	non-web libraries and frameworks
	sahat/satellizer	TypeScript	101,984	129	6	web libraries and frameworks
	winjs/winjs	JavaScipt	252,912	40	6	web libraries and frameworks
Exponential Growth	ionic-team/ionic-framework	TypeScript	193,047	385	7	non-web libraries and frameworks
	scalaz/scalaz	Scala	47,027	184	10	non-web libraries and frameworks
	donnemartin/system-design-primer	Python	10,382	103	3	web libraries and frameworks
	synrc/n2o	Erlang	5,199	60	7	system software
	tmux/tmux	C	58,833	5	5	application software
	sbt/sbt	Scala	78,067	229	11	software tools
Linear Growth	ubernetes/kubernetes	Go	4,199,364	2,781	6	software tools
	rails/rails	Ruby	347,581	4,253	5	software tools
	symfony/symfony	PHP	803,528	2,153	10	web libraries and frameworks
	definitelytyped/definitelytyped	TypeScript	2,586,278	11,763	8	documentation
	alpaca-lang/alpaca	Erlang	13,513	16	4	software tools
	libuv/libuv	C	72,501	395	7	system software

TABLE IV
USE-CASES USED FOR GENDERMAG EVALUATIONS

#	Use-case
UC1: Use-case 1	Find an issue to work on
UC2: Use-case 2	File an issue
UC3: Use-case 3	Make a documentation contribution

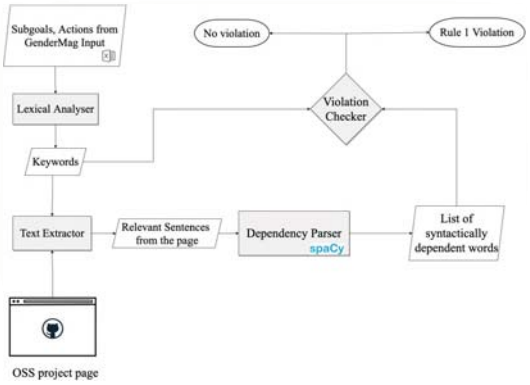

Fig. 5. AID's approach for capturing inclusivity bugs (Rule 1)

example, the wording of the subgoal serves as the information that Abi seeks, and the words from actions serve as cues to direct Abi to a UI action. Without such cues, Abi would face difficulty finding all the information they need. For example, for Project4, UC1 (Find an issue to work on), S1 (Find the issue list)'s Action1, the human team reported, Project4-UC1S1A1: *"There's nothing about the 'issue list'. 'Issue' is mentioned in one of the sections but in a different context"*. Rule 1 captures this using the model shown in Figure 5.

Rule 1: Keywords from subgoals and associated actions should be present on the webpage.

To check if the webpage includes the words from a subgoal and action sequence, AID extracts lexical features, including the Part-of-Speech (POS) tag of each word using spaCy [45]. It then adds the nouns and adjectives from the POS to a list of "keywords" that it searches for in the webpage. For example, in the third row of Table V, Abi's subgoal (S2) is to "File a new bug report issue" and the associated action (A1) is: "Click on the tracker link". AID parses these sentences to add ["new", "bug", "report", "issue"] and ["tracker", "link"] to the list of keywords.

The tool does not include verbs (e.g., "click"), since these are usually commands to a human, not labels or content-oriented keywords. We also exclude DOM words (e.g., link, header, footer, etc.) and words like "information" and "data" from the list of keywords because these words can have overly broad interpretations.

The next step is to look for these keywords in the webpage. Our initial approach performed a simple string search. However, this resulted in keyword matches that were on parts of the webpage that were irrelevant for the specific subgoal.

Thus, to detect relevant information to the specific subgoal and action under evaluation, we retrieve all the sentences on the webpage that contain any of the keywords. Then we use dependency parsing [44] to extract the relationship between the words in each sentence as shown in Figure 6. The term subtree refers to smaller syntactic units within these sentences.

AID then navigates the dependency parse tree (as shown in Figure 6) by extracting the subtree of each of the keywords in them. If one keyword is found in another keyword's subtree, it shows a syntactic relationship between them and is considered relevant to the subgoal or action.

For example, Figure 6 shows the tree structure from a sentence in the webpage of Project-6: "Before submitting a pull request, we ask that you please create an issue that explains the bug or feature request". This results in a dependency parse tree with "ask" as the root. We check the subtree of "issue" and see that "bug" (another keyword) is in the list of elements of the tree. Therefore, AID determines that this sentence is indeed about issues that are bugs and does not report an inclusivity bug when evaluating the action (A1) for subgoal (S2) in Table V.

TABLE V
EXAMPLE OF INPUT TO AID

Use-Case	Subgoal	Action	URL
UC2: File an issue	S1: Find information about filing an issue	A1-Q#1: Click on reporting a bug	symfony.com/doc/current/contributing/code/index.html
UC2: File an issue	S1: Find information about filing an issue	A1-Q#2: Click on reporting a bug	symfony.com/doc/current/contributing/code/bugs.html
UC2: File an issue	S2: File a new bug report issue	A1-Q#1: Click on the tracker link	symfony.com/doc/current/contributing/code/bugs.html
UC2: File an issue	S2: File a new bug report issue	A1-Q#2: Click on the tracker link	github.com/symfony/symfony/issues

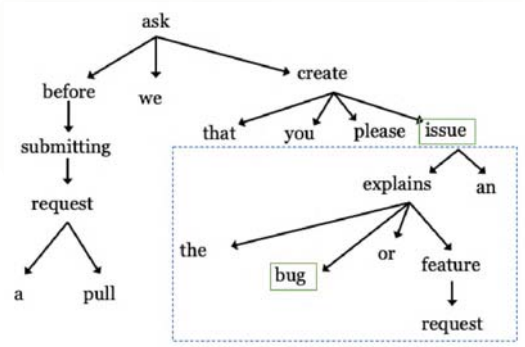

Fig. 6. The dependency parse tree for the sentence "*Before submitting a pull request, we ask that you please create an issue that explains the bug or feature request.*" The dashed blue border encompasses all the elements in the subtree of the keyword "issue". The keywords "issue" and "bug" are bordered in green.

2) Situating Abi in the context of the action performed: On clicking a link, the destination page should offer cues to help Abi's understand that they have reached the right place. If a project page fails to use words similar to what a link label hinted at, OSS'ers like Abi could get confused. For example, when the team clicked on a link labeled "How to file an issue" but didn't find relevant information on the resulting page, their inclusivity bug report said: Project1-UC2S1A3: "*The link takes to a page that does not really mention how to file issues which is what the link said. Abi might not think that this is where she wanted to go.*"

This team's observation echoes others' recommendations that words in a link's label should be prominent on the link's destination page [46], leading to our second rule:

Rule 2: Linked pages should contain keywords from link labels.

AID checks for Rule 2 for linked pages and thus, the pages that are an input to Action Question #2 (e.g., .../issues in S2:A1-Q#2 in Table V). But since it evaluates each page independently (it is agnostic of past CW steps) it needs to identify the link in the previous step that brought the OSS'er to this page. Therefore, it analyzes the webpage that was an input to the previous CW step (S2A1-Q#1, .../bugs.html). It also parses the text of S2A1-Q#1 and using the same parsing technique explained above extracts the nouns to create a list of keywords ("tracker" is the only noun in our example). Next it extracts all the link labels in this page (S2A1-Q#1, .../bugs.html) that match the nouns ("tracker"). Note: this a step to remove nouns from (sometimes wordy) action text that do not match the link label. AID then searches for the

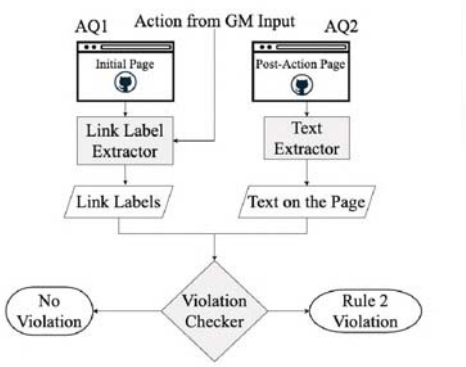

Fig. 7. AID's approach for capturing inclusivity bugs (Rule 2)

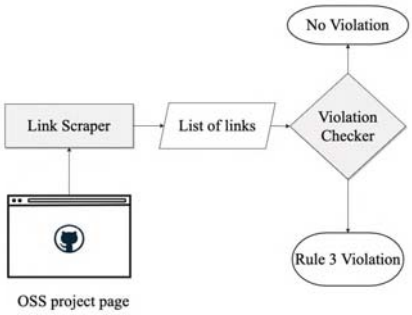

Fig. 8. AID's approach for capturing inclusivity bugs (Rule 3)

keywords in the linked page (i.e., S2A1-Q#2, .../issues). If it cannot find that keyword anywhere in the page it reports an inclusivity bug. Figure 7 details the steps in which AID captures inclusivity bugs arising due to a Rule 2 violation.

3) Information link navigation: OSS'ers like Abi click on a link only after gathering enough information and planning their next step. Labeled links provide Abi with information about the webpage they are supposed to visit. As a team reported for Project15-UC1S1A1: "*It [link] doesn't talk about issue lists and there are a lot of links which don't say where they lead to.*" The team felt that Abi would not know which link to navigate to since all the links in the page had non-descriptive labels.

Other work agrees, recommending that links should have descriptive yet unique link labels and begin with keywords [47]. Rule 3 captures this tenet.

Rule 3: Links should be labeled with a keyword or phrase.

AID checks if links on the page it is evaluating are labeled with a keyword or phrase. To extract the link labels, we wrote a custom web scraper using BeautifulSoup [48]. AID then checks the link labels and reports a bug if the URL of the link is the same as its label, as shown in Figure 8.

4) Cues about "issue" characteristics: Our fourth and fifth rules focused on issues in GitHub. Past research has reported that finding a task to work on is especially difficult for some OSS'ers, such as newcomers [49] and mentors [50]. To address this problem, GitHub recommends using issue labels and provides a set of default labels (e.g., "bugs", "documentation", "good first issue"). Figure 10 shows examples of issue labels.

When OSS'ers like Abi come to the issue page they look for cues from the issue labels to gather information about the task (e.g., the type of task, which part of the codebase are related to the issue, what skills are needed). When issues are not labeled they can get discouraged. As pointed out for Project7-UC1S3A1: *"Neither issue seems to be...very clearly labeled. So because of her info processing style, Abi might not be able to gather enough information about the issues listed and give up*". This leads to our fourth rule (see Figure 9).

Rule 4: Issues should have labels.

AID checks for labels on open issues and reports a bug if the issues are unlabeled. It checks for labels for the first 25 open issues[1], which it extracts using the GitHub API.

However, simply having labeled issues might not be sufficient. The Cognitive Walkthrough at the core of GenderMag is about learnability to a first-time user—i.e., an OSS newcomer in this domain. Thus, the humans' GenderMag sessions reported issues relating to newcomers, such as:

Project6-UC1S3A1: *"The issues are well labeled, but there is no sign as to what would be a good one for a first timer like Abi, she might feel unsure and not choose one."*

Rule 5: Issues should have newcomer-friendly labels (when appropriate).

Rule-5 evaluates the issue label text to check if it is newcomer-friendly by string-matching against the list of newcomer-friendly labels in "MunGell/awesome-for-beginners" GitHub repository [51]. This list is curated from 187 projects in 22 programming languages.

We implemented this set of decision rules with some trepidation. They are text-processing rules done statically, and derived from GenderMag sessions of a relatively small set of projects. Would such rules be able to find the same kinds of inclusivity bugs in arbitrary OSS pages that humans bring to GenderMag sessions by stepping into the shoes of a persona?

V. AID's Effectiveness

We ran AID on the 20 projects described above which produced 353 inclusivity bugs. Compared with the inclusivity bugs identified by the humans who had conducted GenderMag sessions manually on the same projects and use-cases, AID's precision was 0.69, recall was 0.92, and F-measure was 0.79. Table VI details precision, recall, and F-measures.

[1]GitHub shows 25 issues per page and an OSS'er newcomer like Abi is unlikely to look further if she doesn't see any cues for her task.

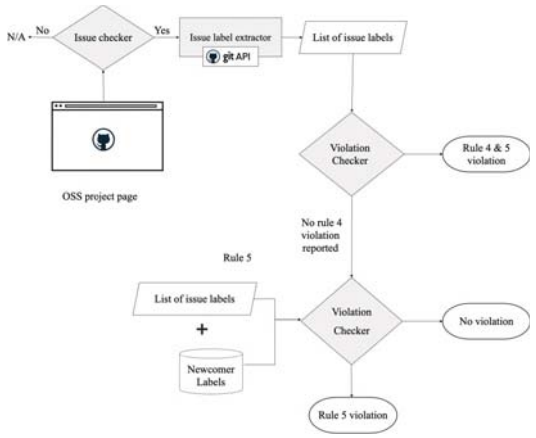

Fig. 9. AID's approach for capturing inclusivity bugs (Rule 4 & 5)

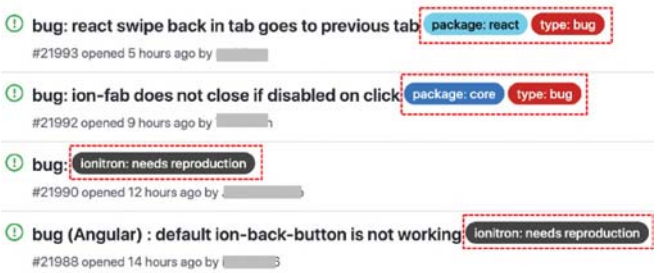

Fig. 10. Example of an inclusivity bug due to Rule 5 violation : Issue labels are not newcomer-friendly

A. Precision: A closer look at false positives

AID disagreed with the human evaluators in 110 of the 353 inclusivity bugs it reported. Since false positives play a critical role in developers' dissatisfaction with current tools [52], AID's precision, a measure of false positive rate, could be critical to its acceptance. In this section, we consider which of these 110 "extra" inclusivity bugs—i.e., those the tool found but the humans did not—really were false positives, and why.

AID indeed produced false positives for three reasons: (1) its use of static analysis instead of dynamic analysis, (2) its reliance solely on the HTML of the webpage, and (3) its semantic limitations. There were 49 of these false positives (45% of the total extra inclusivity bugs).

TABLE VI
AID's Precision, Recall and F-measure, with human GenderMag sessions as the gold standard (Section IV-B).

	Precision	Recall	F-measure
Rule 1	0.64	0.89	0.74
Rule 2	0.75	1.00	0.86
Rule 3	0.70	1.00	0.83
Rule 4	0.74	1.00	0.85
Rule 5	0.53	1.00	0.69
Overall	0.69	0.92	0.79

TABLE VII
61 INCLUSIVITY BUGS WHERE AID WAS CORRECT AND THE HUMAN TEAMS WERE INCORRECT.

Category of Evidence	#	Example
Internal (Team activities):		
Team mentioned the bug later	27	No information about contributions on the README file. AID reported this bug earlier than a team did, but the team found the bug in the next step.
Team didn't mention the bug, but they fixed it later	3	Unlabelled issues on an issue list. Team didn't report it, but we could see from the post-fix project version they had fixed it.
Team mentioned the bug earlier but not here	20	Tool reported a bug every time it found it, but teams did not bother to repeat.
External (Nielsen's)	11	Unlabelled web link was not in the direct path a team was evaluating.

Static Analysis: Analyzing an action like "Make a change to the Readme.md" requires (human) input during the session's walkthrough, and that human input affects what gets displayed next during the walkthrough. Because AID uses solely static analysis of the webpage HTML, it cannot take user inputs like the above into account, which can generate false positives. Addressing this type of false positive would require some way of gathering user input data, either "on the spot" or from a corpus of user inputs to this question. There were three instances of this type of false positives.

Reliance on text content/labels: AID does not currently do image analysis. For example, for the action "click on pencil icon" on the Readme page, AID looks for keywords "pencil" and "icon". If the image filename for the pencil icon is something like word "pencil" or "icon" (e.g., <IMG src="Icons/pencil.png">), AID would realize that it relates to the action; otherwise it does not. In contrast, the human evaluators easily recognized the icon from its appearance on their screen. Addressing this type of false positive would require adding some form of image processing. There were 15 false positives of this type.

English semantics: AID does not currently address underlying semantics of English language usage. For example, "Find something to work on" would translate to "find an issue to work on" for many OSS'ers, but the tool does not know this. Some of AID's semantic limitations could be improved though the use of synonym dictionaries or methods for detecting term similarity like Latent-Semantic Indexing or TF-IDF [53], [54]; others would require semantic analyses accounting for antecedents, referents, and/or context [55]. There were 31 false positives from the tool's semantic limitations.

AID was actually right. The remaining 61 of the 110 "extra" inclusivity bugs were not false positives. Rather, the tool was correct in the three categories: (1) evaluator false negatives, (2) repetitions of the same inclusivity bug, and (3) the tool going "beyond the call of duty" to find things it didn't need to find. Table VII summarizes them. We identified the instances where AID was correct in the following manner. First, we looked at every bug AID found that the GenderMag teams did not. When evidence existed that AID was "right", we associated the evidence with the instances of those bugs, as discussed next.

As Mahotody et al.'s survey of Cognitive Walkthroughs shows, although humans performing methods in the CW-family report few false positives (10% or below across numerous studies), their propensity for false negatives has run as high as 70% false negative rates [25], in which the humans simply miss seeing some of the problems. Likewise with GenderMag, although humans' false positive rates are well below 10% [3], [7], human GenderMag users tend to miss some inclusivity bugs. The human evaluators in this investigation were also subject to this phenomenon. In 27 instances (see Table VII, Row2), the tool flagged a bug that the team missed, but the *team mentioned the bug later* in subsequent steps of their GenderMag sessions.

In another three inclusivity bugs that AID found, the Project-F *team didn't mention the bug, but they fixed it later* as part of their overall changes (see Table VII, Row3). This suggests that the Project-F team eventually realized that it was problematic. From this we conclude that tool was right in identifying these three bugs. Thus, the total of human-tool team differences due to humans' false negatives was 30.

There were 20 instances where AID identified bugs that humans did not report because of repetitions. AID does not realize when it reports the same bug more than once, such as if the same bug arises in more than one step of a use-case. In contrast, humans do realize it, and sometimes do not bother to report the same inclusivity bug if it occurs a second time. In 20 such cases, the *team mentioned the bug earlier, but not here* (see Table VII, Row4). These repetitions can be potentially annoying, but they are not false positives.

Finally, sometimes AID reported inclusivity bugs that were unlabeled links in locations not directly in the direct path of the use-case (see Table VII, Row5). There would have been no reason for the human teams to find these inclusivity bugs if they strictly followed the use-cases. Still, that does not mean that such bugs were false positives. We argue they were true positives: that unlabeled links can be inclusivity bugs for comprehensive information processors like Abi, who want to find all the relevant information. Without a clear label, Abi's would be unable to predict whether information might lie behind the link. This may be why the *Nielsen/Norman group* emphasizes the need for links to use descriptive anchor text that uses keywords to facilitate information processing [47]. The tool's "beyond the call of duty" to relax the strict boundaries of the use-case sequence resulted in 11 of these inclusivity bugs that the human teams did not find.

B. Recall: A closer look at false negatives

AID's recall rate was extremely high, with a total recall of 0.92. In Rules 2-5, AID's recall was a perfect 1.0, perhaps because these rules implement straightforward string comparisons. The tool noticed all instances of rules like these, so whenever the humans noticed it, AID would too.

The tool's recall gaps all came from Rule 1, which produced 20 false negatives. All of these came from the tool's treatment of subgoals' and actions' use of verbs. We decided that, when a word was being used as a verb (e.g., "click" in "click on reporting a bug" in UC2S1A1, Table V) the tool would not treat them as keywords to be searched for in the webpage.

TABLE VIII
TRIANGULATION OF BUGS FOUND BY TEAMS F & J, THE 18 GENDERMAGS WE CONDUCTED, AND AID. OVER 70% OF THE HUMAN-REPORTED BUGS HUMAN FELL INTO F & J'S ORIGINAL 9 CATEGORIES. AID FOUND 74% OF THE BUGS THE HUMANS FOUND.

Inclusivity Bugs	F	J	18 Projects	AID
Not enough information: page	✓		38	32
README: Unclear path to the readme	✓	✓	17	17
README: no info about subgoal	✓		39	33
Lacks understanding of OSS terms: can't match to subgoal	✓✓		5	5
Template: Pull request: not enough instructions		✓✓	1	1
Template: Filing an issue: not enough instructions	✓	✓	23	2
Not enough information: to choose options	✓	✓✓	12	11
Not enough information: to take action	✓✓✓✓		17	15
ISSUES in issue list: not enough information for users to pick a task	✓	✓	26	17
Other			79	70
Total	12	7	257	203

However, in the subgoal (S1): "Find information about how to file an issue", the verb "file" describes the process of reporting an issue. Thus, although the tool did not consider the word "file" to be worth looking for here, the human teams in their sessions did look for information about the process of filing an issue on the project page: Project3-UC2S1A1"*...there is nothing that says [about] file a new issue so Abi might feel some uncertainty with her action.*" This issue accounted for all 20 of AID's false negatives.

C. Results Triangulation

Table VIII shows the inclusivity bugs reported by all sources in the categories of bugs the humans found, triangulated among Project F, Project J, the 18 sessions we conducted, and AID. Each bug category occupies a row of the table. (The table does not include the bugs that AID found that the humans did not.) AID's low performance on Template bugs was due to its static-only analysis, as discussed earlier. Every category of bug was cross-validated by at least three of these four sources and, as the table shows, AID found 74% (203/(12+7+257)) of the bugs the humans reported.

VI. THREATS TO VALIDITY

This section presents the different threats to validity and how we mitigated them.

First, two teams of researchers manually identified the inclusivity bugs using the GenderMag method, so subjectivity of the data can be considered a threat to validity. To minimize this threat, we calculated the inter-rater reliability among the two teams and also validated our data against inclusivity bugs found by two external OSS teams (F & J) using GenderMag.

The second threat to validity of our tool lies in the 110 disagreements between AID's findings and human evaluators' findings. In order to mitigate that, we further analyzed these disagreements to understand their origin. We found different types of disagreements, some were linked to the limited scope of our tool (when the tool was wrong), others (where the tool was more comprehensive than a human evaluator) originated from humans not capturing all possible inclusivity issues.

The final threat is the generalizability of AID in its current scope. AID can be applied to any OSS project in GitHub (28 million public repos) and other hosting sites (Gitlab, bitbucket). As a first step, AID focused on OSS Projects hosted on GitHub, thus, the applicability of our tool might not generalize to other version control platforms or other non-OSS technology. However, some of AID's rules (e.g., Rule1) can be applied on any web app or webpage. AID also currently automates only a vertical slice of GenderMag, with the Abi persona and their specific information processing style. This scoping allowed us to investigate AID's feasibility and effectiveness under stricter controls, but impacts its generalizability.

Most of these threats can be mitigated by (1) future work that addresses the limitations of the tool; (2) extending AID to also evaluate for Abi's other cognitive facets, and to the other GenderMag personas (Pat, Tim); and (3) expanding the scope of software upon which AID runs beyond OSS platforms.

VII. RELATED WORK

The nearest neighbours of the GenderMag method are the Williams recommendations and InclusiveMag. Williams pointed out several ways gender-inclusivity bugs can arise, such as hidden gender bias during the product cycle and women being reluctant to voice their opinions when they are outnumbered in a brainstorming session [16], [56]. Williams also offers concrete recommendations to head off such causal factors, such as assembling groups of at least 60% women during brainstorming sessions and having an equal vote distribution by gender when using informal voting systems [16]. InclusiveMag [17] is a meta-method that spawns methods like GenderMag, for under-served software users. It provides a step-by-step approach for researchers and practitioners to generate methods to evaluate their software. However, neither of these methods have been automated.

GenderMag is a member of the Cognitive Walkthrough family. Mahatody et al.'s [25] comprehensive literature survey of CWs describes many CW variations, some of which focus on reducing problems with the classic CW [57]–[59] such as by reducing the time it requires. Nielsen et al. recommended that a note-taking tool for CWs to address issues like these by guiding the analyst through each CW step, in order to avoid missing steps and to more accurately record results, integrating a CW tool into a prototyping tool [60]. When CWs were first introduced, Rieman et al. created a tool to record the results of a human-run CW [61]. The GenderMag Recorder's Assistant [62] is a recent note-taking tool to help humans organize GenderMag session output—their answers to the CW questions, additional notes and screenshots pertinent to each question. These tools are for supporting humans doing a CW and not the automation of the method itself.

Farther afield, there is a variety of work relating to automation of usability evaluation. For example, Mathur et al. introduced a "Usability Evaluation Framework" [63], a model to automate usability evaluation for mobile apps based on

prevailing usability guidelines. Baker et al. created a prototype to automate usability testing for handheld devices [64]. It measures software usability by recording user actions and compares the user's actions to those expected by the developer. The WebTANGO prototype [65] predicts user's information seeking behaviour and webpage navigations, calculating factors like the time for a user to scan a page, based on its complexity. Dingli et al. proposed a framework and tool that evaluates websites by considering usability guidelines [66]. However, tools like these do not evaluate inclusivity.

Various tools exist to help with accessible design. WAVE [67] is a tool for evaluating accessibility principles. It indicates if information on a webpage is accessible by visually impaired people by checking for alternative text, structural markup, and reading order. Additionally, it flags audio content for the deaf population. Vischeck [68] is a tool for low-vision simulation, which uses image processing techniques to create views for low-vision users. AATT [69] provides an accessibility API to test web applications for conformance to the Web Content Accessibility Guidelines (WCAG). Even though these tools assist in the implementation for more inclusive products, they do not consider cognitive diversity.

Closest to our work is the AutoCWW [70], a tool that automates the CW to identify website navigation problems [71]. It takes a goal from the user along with a list of webpages the user is supposed to navigate. It then computes the similarity of the goal text with the headings and link labels of the input pages, to see if a user who wanted to find the right link could navigate to the correct webpage. AutoCWW only evaluates webpages with respect to the overall use-case while AID has a more fine-grained task oriented approach. AutoCWW assumes a "generic" user and not inclusivity of diverse users. For example, it does not take into account different styles of processing information such as Abi's comprehensive information processing style. GitHub OSS'ers with this style are unlikely to just click on potentially suitable links until after gathering enough information to understand what is available and how they might like to proceed through it. In contrast, AID checks if a user *like Abi* (and, in future versions, like Tim or like Pat) would gather the "right amount" of information to navigate in the needed direction for their current actions.

VIII. Concluding Remarks

The effectiveness that AID showed at automatically detecting these 20 OSS project sites' inclusivity bugs is promising. As we have pointed out, the five rules at the core of our model used static, text-processing techniques. Despite this fact, even under the conservative assumption that humans were the "gold standard", AID achieved a precision rate of 0.69. Further, closer scrutiny revealed that more than half of its false positives under this assumption were not actually false positives (Table VI). Further, its recall rate of 0.92 is better than most recall rates by human users of the CW family [25].

How could the tool have done so well? Researchers have reported the heavy workloads some humans experience using GenderMag [72]. Part of their workload is trying to "become" someone else (here, one of the GenderMag personas), which some humans have difficulty doing (e.g., [35]). Yet, the tool did remarkably well with static text analysis, without even attempting to model the persona.

We hypothesize that the reason for AID's success lies in the concrete, pattern-based approach we used to develop the rules. When humans use GenderMag, they work from the *root* of inclusivity bugs—problem-solving style diversity. In contrast, in developing the rules, we worked from patterns of *symptoms* of the problems the human teams found, and encoding them in rules. Our results suggest that the concrete, pattern-based approach we used is promising.

A goal of this work was to find out whether it was possible for a software tool to automatically detect inclusivity bugs. As our results with AID show, such a tool is both possible and feasible to implement.

Despite its concrete rules, AID is rooted in theory, inherited from the foundations of GenderMag. As Shaw and others have argued, scientific theory lets technological development pass limits previously imposed by relying on intuition and experience [73]. For example, Shaw points out, for centuries, many architectural structures (buildings, bridges, tunnels, canals) could be built only by master craftsmen. Not until scientists developed theories of statics and strength of materials, were today's extraordinary engineering accomplishments possible, such as the Hong Kong–Zhuhai–Macau Bridge spanning 55 miles of ocean, by "ordinary" engineers. In computer science we see the same phenomenon. For example, expert developers once built compilers using only hard-won intuitions gained from extensive experience, but formal language theory has brought tasks like parser and compiler writing to a level where undergraduate computer science students now routinely build them in their coursework [74]. AID follows this path through its theory foundations, as a step toward enabling "ordinary" developers to find inclusivity bugs without needing to become experts in GenderMag.

Our survey's results suggest that tools like this are needed: 92% of our respondents lacked any specific tools or approaches they could use to check for inclusivity bugs.

> P115: *"I wasn't aware of methods, but I try to get feedback from and build for a diverse set of users."*
>
> P156: *"I can make mistakes ...[but] I strive to not do evil."*

IX. Acknowledgements

We thank our survey participants and Teams F & J. This work is partially supported by the National Science Foundation under Grant numbers 1815486, 1901031 and 21-0045.

References

[1] M. M. Burnett, L. Beckwith, S. Wiedenbeck, S. D. Fleming, J. Cao, T. H. Park, V. Grigoreanu, and K. Rector, "Gender pluralism in problem-solving software," *Interacting with computers*, vol. 23, no. 5, pp. 450–460, 2011.

[2] L. Beckwith, C. Kissinger, M. Burnett, S. Wiedenbeck, J. Lawrance, A. Blackwell, and C. Cook, "Tinkering and gender in end-user programmers' debugging," in *Proceedings of the SIGCHI conference on Human Factors in computing systems*, 2006, pp. 231–240.

[3] M. Burnett, S. Stumpf, J. Macbeth, S. Makri, L. Beckwith, I. Kwan, A. Peters, and W. Jernigan, "Gendermag: A method for evaluating software's gender inclusiveness," *Interacting with Computers*, vol. 28, no. 6, pp. 760–787, 2016.

[4] M. Burnett, R. Counts, R. Lawrence, and H. Hanson, "Gender hcl and microsoft: Highlights from a longitudinal study," in *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2017, pp. 139–143.

[5] M. Burnett, S. D. Fleming, S. Iqbal, G. Venolia, V. Rajaram, U. Farooq, V. Grigoreanu, and M. Czerwinski, "Gender differences and programming environments: across programming populations," in *Proceedings of the 2010 ACM-IEEE international symposium on empirical software engineering and measurement*, 2010, pp. 1–10.

[6] D. Showkat and C. Grimm, "Identifying gender differences in information processing style, self-efficacy, and tinkering for robot teleoperation," in *2018 15th International Conference on Ubiquitous Robots (UR)*. IEEE, 2018, pp. 443–448.

[7] M. Vorvoreanu, L. Zhang, Y.-H. Huang, C. Hilderbrand, Z. Steine-Hanson, and M. Burnett, "From gender biases to gender-inclusive design: An empirical investigation," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–14.

[8] R. Tatman, "Gender and dialect bias in YouTube's automatic captions," in *Proceedings of the First ACL Workshop on Ethics in Natural Language Processing*. Valencia, Spain: Association for Computational Linguistics, Apr. 2017, pp. 53–59. [Online]. Available: https://www.aclweb.org/anthology/W17-1606

[9] C. Pérez and L. Sáenz, "Gender-inclusive language in games and its localization challenges," https://medium.com/@keywordsstudios/gender\-inclusive\-language\-in\-games\-and\-its\-localization\-challenges\-e132fde36b76, 2019, accessed: 2020-08-27.

[10] S. Stumpf, A. Peters, S. Bardzell, M. Burnett, D. Busse, J. Cauchard, and E. Churchill, "Gender-inclusive hci research and design: A conceptual review," *Foundations and Trends in Human-Computer Interaction*, vol. 13, no. 1, pp. 1–69, 2020.

[11] X. Kondrat, "Gender and video games: How is female gender generally represented in various genres of video games?" *Journal of Comparative Research in Anthropology and Sociology*, vol. 6, no. 01, pp. 171–193, 2015.

[12] I. A. Hamilton, "Why it's totally unsurprising that amazon's recruitment ai was biased against women," https://www.businessinsider.com/amazon-ai-biased-against-women-no-surprise-sandra-wachter-2018-10, 2018, accessed: 2020-08-27.

[13] M. Kay, C. Matuszek, and S. A. Munson, "Unequal representation and gender stereotypes in image search results for occupations," in *Proceedings of the ACM Conference on Human Factors in Computing Systems*, 2015, pp. 3819–3828.

[14] M. Burnett, A. Peters, C. Hill, and N. Elarief, "Finding gender-inclusiveness software issues with gendermag: a field investigation," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 2016, pp. 2586–2598.

[15] S. H. Padala, C. J. Mendez, L. F. Dias, I. Steinmacher, Z. S. Hanson, C. Hilderbrand, A. Horvath, C. Hill, L. Simpson, M. Burnett, M. Gerosa, and A. Sarma, "How gender-biased tools shape newcomer experiences in oss projects," *IEEE Transactions on Software Engineering*, 2020.

[16] G. Williams, "Are you sure your software is gender-neutral?" *Interactions*, vol. 21, no. 1, pp. 36–39, 2014.

[17] C. Mendez, L. Letaw, M. Burnett, S. Stumpf, A. Sarma, and C. Hilderbrand, "From gendermag to inclusivemag: An inclusive design meta-method," in *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2019, pp. 97–106.

[18] S. J. Cunningham, A. Hinze, and D. M. Nichols, "Supporting gender-neutral digital library creation: A case study using the gendermag toolkit," in *International Conference on Asian Digital Libraries*. Springer, 2016, pp. 45–50.

[19] A. Shekhar and N. Marsden, "Cognitive walkthrough of a learning management system with gendered personas," in *Conference on Gender & IT*. ACM, 2018, pp. 191–198.

[20] C. Hilderbrand, C. Perdriau, L. Letaw, J. Emard, Z. Steine-Hanson, M. Burnett, and A. Sarma, "Engineering gender-inclusivity into software: Ten teams' tales from the trenches," in *ACM/IEEE International Conference on Software Engineering*, 2020.

[21] C. Mendez, H. S. Padala, Z. Steine-Hanson, C. Hilderbrand, A. Horvath, C. Hill, L. Simpson, N. Patil, A. Sarma, and M. Burnett, "Open source barriers to entry, revisited: A sociotechnical perspective," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 1004–1015.

[22] M. Burnett, S. Stumpf, L. Beckwith, and A. Peters, "The gendermag kit: How to use the gendermag method to find inclusiveness issues through a gender lens," 2020.

[23] C. Gralha, M. Goulao, and J. Araujo, "Analysing gender differences in building social goal models: A quasi-experiment," in *IEEE International Requirements Engineering Conference*, ser. RE 2019, 2019.

[24] J. Meyers-Levy and B. Loken, "Revisiting gender differences: What we know and what lies ahead," *Journal of Consumer Psychology*, vol. 25, no. 1, pp. 129–149, 2015.

[25] T. Mahatody, M. Sagar, and C. Kolski, "State of the art on the cognitive walkthrough method, its variants and evolutions," *Intl. Journal of Human–Computer Interaction*, vol. 26, no. 8, pp. 741–785, 2010.

[26] Anonymous, "AID: An automated inclusivity-bug detector," Aug. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.4007579

[27] K. Blincoe, J. Sheoran, S. Goggins, E. Petakovic, and D. Damian, "Understanding the popular users: Following, affiliation influence and leadership on github," *Information and Software Technology*, vol. 70, 10 2015.

[28] G. user s0md3v. (2020) Zen find email addresses of github users. [Online]. Available: https://https://github.com/s0md3v/Zen

[29] F. Medeiros, M. Ribeiro, R. Gheyi, S. Apel, C. Kästner, B. Ferreira, L. Carvalho, and B. Fonseca, "Discipline matters: Refactoring of preprocessor directives in the ifdef hell," *IEEE Transactions on Software Engineering*, vol. 44, no. 5, pp. 453–469, 2018.

[30] M. Burnett, S. D. Fleming, S. Iqbal, G. Venolia, V. Rajaram, U. Farooq, V. Grigoreanu, and M. Czerwinski, "Gender differences and programming environments: across programming populations," in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 2010, pp. 1–10.

[31] M. Burnett, "Womenomics and gender-inclusive software: What software engineers need to know (invited talk)," in *ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016.

[32] R. Grey, "Inclusive chromium code." [Online]. Available: https://chromium.googlesource.com/chromium/src/+/master/styleguide/inclusive_code.md

[33] Google, "Writing inclusive documentation." [Online]. Available: https://developers.google.com/style/inclusive-documentation

[34] *Deep Learning for Coders with fastai and PyTorch*. O'Reilly, 2020.

[35] A. Oleson, C. Mendez, Z. Steine-Hanson, C. Hilderbrand, C. Perdriau, M. Burnett, and A. J. Ko, "Pedagogical content knowledge for teaching inclusive design," in *Proceedings of the 2018 ACM Conference on International Computing Education Research*, 2018, pp. 69–77.

[36] M. H. Blackmon, P. G. Polson, C. Lewis *et al.*, "Automated cognitive walkthrough for the web (autocww)," in *Workshop Date: April*, vol. 21, 2002, p. 22.

[37] A. Bandura, *Social Foundations of Thought and Action*. Prentice Hall, 1986.

[38] F. Fronchetti, I. Wiese, G. Pinto, and I. Steinmacher, "What attracts newcomers to onboard on oss projects? tl; dr: Popularity," in *IFIP International Conference on Open Source Systems*. Springer, 2019, pp. 91–103.

[39] S. E. Stemler, "A comparison of consensus, consistency, and measurement approaches to estimating interrater reliability," *Practical Assessment, Research, and Evaluation*, vol. 9, no. 1, p. 4, 2004.

[40] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *biometrics*, pp. 159–174, 1977.

[41] D. R. Garrison, M. Cleveland-Innes, M. Koole, and J. Kappelman, "Revisiting methodological issues in transcript analysis: Negotiated coding and reliability," *The Internet and Higher Education*, vol. 9, no. 1, pp. 1–8, 2006.

[42] J. Pustejovsky and B. Boguraev, "Lexical knowledge representation and natural language processing," *Artificial Intelligence*, vol. 63, no. 1-2, pp. 193–223, 1993.

[43] S. Abney, "Part-of-speech tagging and partial parsing," in *Corpus-based methods in language and speech processing*. Springer, 1997, pp. 118–136.

[44] J. Nivre, "An efficient algorithm for projective dependency parsing," in *Proceedings of the Eighth International Conference on Parsing Technologies*, 2003, pp. 149–160.

[45] *spaCy Natural Language Processing Library*, 2015 (accessed April, 2020), https://spacy.io/.

[46] K. Pernice, *A Link is a Promise*, 2014 (accessed June, 2020), https://www.nngroup.com/articles/link-promise/.
[47] M. McCloskey, *Writing Hyperlinks: Salient, Descriptive, Start with Keyword*, 2014 (accessed June, 2020), https://www.nngroup.com/articles/writing-links/.
[48] L. Richardson, *beautifulsoup 4 4.9.1*, 2006 (accessed April, 2020), https://pypi.org/project/beautifulsoup4/.
[49] I. Steinmacher, T. Conte, M. A. Gerosa, and D. Redmiles, "Social barriers faced by newcomers placing their first contribution in open source software projects," in *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work Social Computing*, ser. CSCW '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1379–1392. [Online]. Available: https://doi.org/10.1145/2675133.2675215
[50] S. Balali, U. Annamalai, H. S. Padala, B. Trinkenreich, M. A. Gerosa, I. Steinmacher, and A. Sarma, "Recommending tasks to newcomers in oss projects: How do mentors handle it?"
[51] "Awesome first PR opportunities," https://github.com/MunGell/awesome-for-beginners, 2019, accessed: 2019-08-17.
[52] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?" in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 672–681.
[53] S. T. Dumais *et al.*, "Latent semantic indexing (lsi) and trec-2," *Nist Special Publication Sp*, pp. 105–105, 1994.
[54] R. Baeza-Yates, B. Ribeiro-Neto *et al.*, *Modern information retrieval*. ACM press New York, 1999, vol. 463.
[55] J. Han, A. Sun, H. Zhang, C. Li, and S. Shi, "Case: Context-aware semantic expansion." in *AAAI*, 2020, pp. 7871–7878.
[56] C. F. KARPOWITZ, T. MENDELBERG, and L. SHAKER, "Gender inequality in deliberative participation," *American Political Science Review*, vol. 106, no. 3, p. 533–547, 2012.
[57] V. Grigoreanu and M. Mohanna, "Informal cognitive walkthroughs (icw) paring down and pairing up for an agile world," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2013, pp. 3093–3096.
[58] A. Sears, "Heuristic walkthroughs: Finding the problems without the noise," *Int. J. Hum. Comput. Interact.*, vol. 9, pp. 213–234, 1997.
[59] R. Spencer, "The streamlined cognitive walkthrough method, working around social constraints encountered in a software development company," in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, 2000, pp. 353–359.
[60] N. Jacobsen and B. John, "Two case studies in using cognitive walkthrough for interface evaluation," 2000.
[61] J. Rieman, S. Davies, D. C. Hair, M. Esemplare, P. Polson, and C. Lewis, "An automated cognitive walkthrough," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '91. New York, NY, USA: Association for Computing Machinery, 1991, p. 427–428. [Online]. Available: https://doi.org/10.1145/108844.108986
[62] C. Mendez, Z. S. Hanson, A. Oleson, A. Horvath, C. Hill, C. Hilderbrand, A. Sarma, and M. Burnett, "Semi-automating (or not) a socio-technical method for socio-technical systems," in *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2018, pp. 23–32.
[63] N. Mathur, S. A. Karre, and Y. R. Reddy, "Usability evaluation framework for mobile apps using code analysis," in *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, 2018, pp. 187–192.
[64] F. T. W. Au, S. Baker, I. Warren, and G. Dobbie, "Automated usability testing framework," in *Proceedings of the Ninth Conference on Australasian User Interface - Volume 76*, ser. AUIC '08. AUS: Australian Computer Society, Inc., 2008, p. 55–64.
[65] M. Y. Ivory, "Web tango: towards automated comparison of information-centric web site designs," in *CHI'00 extended abstracts on Human factors in computing systems*, 2000, pp. 329–330.
[66] A. Dingli and J. Mifsud, "Useful: A framework to mainstream web site usability through automated evaluation," 2011.
[67] L. R. Kasday, "A tool to evaluate universal web accessibility," in *Proceedings on the 2000 conference on Universal Usability*, 2000, pp. 161–162.
[68] *Vischeck*, (accessed Dec, 2020), http://www.vischeck.com/.
[69] *AATT*, (accessed Dec, 2020), https://github.com/paypal/AATT.
[70] M. Kitajima, M. H. Blackmon, P. G. Polson, and C. Lewis, "Autocww : Automated cognitive walkthrough for the web," 2002.
[71] M. Blackmon, P. Polson, M. Kitajima, and C. Lewis, "Cognitive walkthrough for the web," vol. 4, 04 2002, pp. 463–470.
[72] C. Hill, S. Ernst, A. Oleson, A. Horvath, and M. Burnett, "Gendermag experiences in the field: The whole, the parts, and the workload," in *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2016, pp. 199–207.
[73] M. Shaw, "Prospects for an engineering discipline of software," *IEEE Software*, vol. 7, no. 6, pp. 15–24, 1990.
[74] A. Aho, M. Lam, R. Sethi, and J. Ullman, "Compilers: Principles, techniques, and tools . boston, ma, usa: Addison-wesley longman publishing co," p. 1009, 2006.