Software Fairness

Yuriy Brun and Alexandra Meliou
University of Massachusetts, Amherst
Amherst, Massachusetts 01003-9264, USA
{brun, ameli}@cs.umass.edu

ABSTRACT

A goal of software engineering research is advancing software quality and the success of the software engineering process. However, while recent studies have demonstrated a new kind of defect in software related to its ability to operate in fair and unbiased manner, software engineering has not yet wholeheartedly tackled these new kinds of defects, thus leaving software vulnerable. This paper outlines a vision for how software engineering research can help reduce fairness defects and represents a call to action by the software engineering research community to reify that vision. Modern software is riddled with examples of biased behavior, from automated translation injecting gender stereotypes, to vision systems failing to see faces of certain races, to the US criminal justice sytem relying on biased computational assessments of crime recidivism. While systems may learn bias from biased data, bias can also emerge from ambiguous or incomplete requirement specification, poor design, implementation bugs, and unintended component interactions. We argue that software fairness is analogous to software quality, and that numerous software engineering challenges in the areas of requirements, specification, design, testing, and verification need to be tackled to solve this problem.

ACM Reference format:
Yuriy Brun and Alexandra Meliou. 2018. Software Fairness. In *Proceedings of The 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Lake Buena Vista, FL, USA, 4–9 November, 2018 (ESEC/FSE 2018),* 6 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 BIAS IN SOFTWARE

Bias is rampant in modern software. Try it out: type "He is a nurse. She is a doctor." into http://translate.google.com and translate it into Turkish. Then translate the result ("O bir bebek hemşire. O bir doktor.") into English and you get "She is a nurse. He is a doctor." [18] (Figure 1). Want to upload a video to YouTube and have closed captions generated automatically? If your video features a man's voice, the closed captions are likely to be more accurate than one with a woman's voice [65]. Want your futuristic house or autonomous car to recognize you when you get in? Facial recognition systems often perform poorly on female and African American faces [40]. Looking

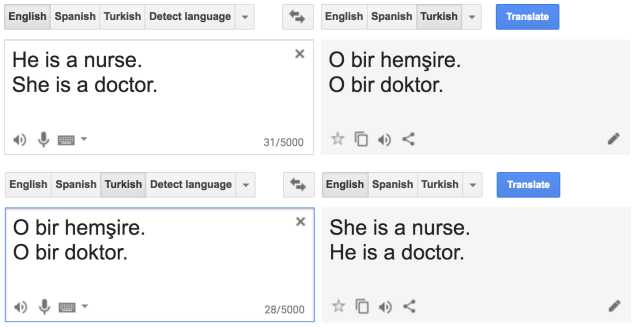

Figure 1: Automatically translating text from a language without gendered pronouns into English can be influenced by societal biases [18].

for a job? Hopefully, when an employer searches for your name, the search engine doesn't decide to show ads about your (nonexistent) arrest record only because of your race [64] (Figure 2). Evidence of bias in software goes on and on. In 2016, Amazon software decided not to offer same-day delivery to predominantly minority neighborhoods [43]. Staples offered online discounts to customers only in wealthier and more affluent neighborhoods [30, 52]. Language processing tools work better on English written by white people than people of other races [8]. The software US courts use to assess the risk of a criminal repeating a crime when deciding on bail and sentencing has been shown to exhibit racial bias [4].

With the increased use of machine learning in business, biases in underlying data used to train such systems can infiltrate the systems' outcomes. The machine learning community is aware of this problem and has begun research on building machine classifiers that are fair in the face of biased data [16, 32, 73, 75], e.g., as part of its Fairness, Accountability, and Transparency in Machine Learning workshops (https://www.fatml.org/). The recently formed Conference on Fairness, Accountability, and Transparency (https://fatconference.org/) brought together, in 2018, researchers

Figure 2: Though since fixed, in 2013, Google searches for traditionally African American names (e.g., "Latanya Sweeney") were more likely to result in ads for arrest records [64].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ESEC/FSE 20184–9 November, 2018Lake Buena Vista, FL, USA
© 2018 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

and practitioners interested in related topics. But while this conference was heavily attended by machine learning, policy, and legal experts, notably, software researchers and engineers were not well represented.

Despite the relatively low involvement, to date, of the software engineering community in this endeavor (with a few notable exceptions [2, 28, 68] discussed in Sections 2.3 and 2.4), the problems of producing fair software are, in fact, squarely in the software engineering domain. Much like other important software properties, such as quality or security, producing fair software requires (1) eliciting and specifying requirements that capture fairness properties, (2) architecting and designing systems while adhering to their fairness concerns, (3) validating and verifying fairness properties of the resulting software products, and (4) maintaining the fairness properties as the software evolves. As with software security and quality, it is insufficient to treat software fairness as an after-the-system-is-built concern; instead, fairness needs to be a first-class entity in the software engineering process, and doing so requires language, tool, and automation support at the levels of requirements, design, implementation, testing, and verification.

This paper represents a call to action for software engineering researchers to tackle the important challenges of engineering fair software. We outline the challenges that must be tackled to support developing fair systems and argue for the urgency of such research. Section 2 lays out the relevant challenges and presents research ideas our community should tackle. Section 3 places our call to action in the context of ongoing research. Finally, Section 4 summarizes our key arguments for the proposed line of research.

2 FAIRNESS RESEARCH CHALLENGES

How do we fight bias in software? Developing better machine learning methods is a start, but because there are many causes of bias, it cannot be the entire solution. Software fairness is analogous to software quality and security: good design and proper algorithms are important, but so are quality control via, e.g., testing and formal verification. For example, today, both companies and customers can use software testing to measure software quality, identify and report bugs, and prevent regressions. Similarly, tools are needed to support software *fairness* testing to measure software discrimination, enable identifying and reporting discrimination bugs, and protect against discrimination regressions — changes to code or data that introduce discrimination.

Experience has shown that high-level software concerns, such as security, quality, maintainability, self-adaptivity, etc. must be treated as first-class entities throughout the development lifecycle — not ignored until after the system is implemented — or else they are unlikely to be satisfied in the final product, e.g., [3, 19]. Fairness is similar, and when fairness is not made a first-class concern, systems result in unexpected, biased behavior, e.g., [18, 28, 40, 65]. This section outlines the research challenges that need to be addressed for software engineers to treat fairness as a first-class concern when building systems. This list is, of course, inherently partial, and more challenges are likely to arise as research progresses.

2.1 Requirements and Specifications

What does it mean for software to be fair? There have emerged numerous definitions of algorithmic fairness [55]. But while each of these definitions is appropriate in a given context, many are impossible to satisfy simultaneously [27]. For example, consider software that decides if an applicant should receive a loan, and consider fairness metrics with respect to race. The group fairness definition [16, 22, 32, 73, 75] states that the same fraction of the applicants of each race should get loans. If a higher fraction of applicants of one race receive loans than of another, that's bias. Meanwhile the individual or causal fairness definition [22, 28] states that no two applicants identical in every way except race should result in one receiving the loan and the other not. So if there are two applicants who differ only in race, they should either both get a loan or neither of them should get a loan. Both of these properties seem desirable in a fair system. However, when applied to the real world, the two definitions cannot be (non-degeneratively, e.g., giving loans to no one) satisfied simultaneously. Suppose there is strong correlation between the applicants' race and income (or any other attribute). Then it is not possible to both give the same fraction of individuals of all races a loan *and* to give all pairs of individuals identical in every way except race a loan. If only everyone above a certain income receives the loan, individual fairness is satisfied but group fairness is not. If the same fraction of applicants of each race receives a loan, then for the race correlated with lower incomes, some individuals who receive a loan must have lower incomes than some individuals of another race who do not receive a loan, and thus individual fairness cannot be satisfied.

The creation of a catalog of fairness definitions and guidelines for when each definition should be used is an open challenge in the engineering of fair software. Of the dozens of existing definitions [55], each depends on and is appropriate for certain contexts. To properly elicit and specify fairness requirements, the software engineers and often their clients must understand these contexts and the relationships between the contexts and the potentially desired fairness properties. Guidelines are needed to drive the questions clients should be asked, concerns and assumptions that need to be considered in selecting fairness definitions to use in specifying requirements, and the implications and restrictions imposed by these requirements.

The creation of requirement consistency and implication analyses is another open challenge. As mentioned earlier, combinations of fairness requirements that rely on multiple definitions of fairness may be, at times counterintuitively, mutually exclusive [27], and automated analyses or simulation can identify inconsistent or unsatisfiable requirements. Further, certain fairness requirements may result in unexpected or undesirable behavior. For example, recent work has shown that on one dataset, placing a constraint on a decision-tree machine-learning system [32] not to discriminate against gender significantly increased its discrimination against race [28]. Such an outcome is likely to be an unwelcome surprise to the software stakeholders who specified the requirement of no bias against gender. Analyses that help understand the possible implications of such constraints and how fairness requirements affect other requirements are critical for understanding the trade-offs and for properly specifying such requirements.

Further, analyses can help understand when satisfying a requirement does not achieve the overall desired goal. For example, the group fairness property described above can be tricked to appear as if the system technically satisfies the property, while failing to satisfy it in spirit. Consider an instance of the loan recommendation system that recommends loans to all tall applicants of race α but to no short applicants of race α, and to all short applicants of race β but to no tall applicants of race β. Despite clear bias with respect to race and height, a requirement of no group discrimination with respect to race is satisfied because discrimination against short applicants of race α is cancelled out by discrimination against tall applicants of race β. This phenomenon can easily be missed when specifying requirements, and automated analyses are needed to mitigate such risks.

2.2 Architecture and Design

Inconsistencies among desired system properties are common. For example, security and usability properties are often at odds, with security making systems harder to use. It is often difficult to tell prior to developing a system architecture or prototype how requirements will interact [62], and the case is likely to be no different for potentially conflicting fairness requirements. An open research challenge is to create tools that help model system architecture and identify potential conflicts or trade-offs. Similar challenges in understanding trade-offs, in terms of functional and non-functional properties of systems, have been tackled with discrete-event simulation [61, 63] and fuzzy logic [23], and such approaches can potentially help with making fairness-related design decisions. Multi-objective optimization can help identify fair trade-offs when considering treating multiple software stakeholders fairly when eliciting requirements from them [24]. Similar approaches could perhaps be used to balance mutually opposing fairness requirements.

Years of system-building experience has produced architectural styles and design patterns for various desirable software properties [66], e.g., the peer-to-peer architectural style to avoid single points of failure. An open research challenge is to develop styles and patterns for fairness properties, perhaps augmenting the above-mentioned catalog of fairness definitions with which styles and patterns help address each definition. Further, style and pattern interactions need to be studied to understand which ones can, and which ones cannot be composed.

For systems that rely on machine learning, the design of fairness-aware machine learning algorithms that can produce fair models even when faced with biased training data is critical. Such algorithms are in relatively early stages of development, e.g., [16, 32, 73, 75], but more mature frameworks with theoretical guarantees are emerging [67].

2.3 Testing and Debugging

The primary method for ensuring software quality is testing [3]. There is strong reason to believe the same should be true for software fairness. Even advanced tools to ensure fairness of requirements and design cannot prevent all bugs, unintended interactions, assumptions that fail to hold in the real world, and other causes of bias. Testing can, as with quality, assess the fairness of the built system, provide a mechanism for debugging bias, and guard against bias regressions.

To date, relatively little software engineering research has focused on these issues, with the notable exceptions of Themis [28**?**] and FairTest [68]. Themis is a mechanism for automated test generation for systems with categorical inputs, such as, for example, the loan recommendation system described above. Themis automatically generates fairness test suites for group and causal (individual) fairness; executing these test suites produces a fairness score and a measure of confidence in that score. FairTest allows developers to write manual tests and performs various analyses to measure fairness-related statistics.

Fairness bugs are common in software systems with complex inputs and outputs, to which Themis and FairTest cannot directly apply. For example, tools with natural English inputs parse English written by white people more accurately than that written by people of other races [8], and facial detection and recognition tools' accuracy also depends on demographic information, such as race and gender [40]. A major challenge for automated fairness test generation is to generate tests for systems with such inputs. Testing such systems for causal fairness requires the generation of carefully controlled natural language and photos that are identical except for a fixed set of sensitive attributes. Further, automated test generation does not yet address covariants in attributes; accounting for such covariants can generate more accurate test suites.

The large feature spaces involved in applications with potential bias create a significant scaling challenge for testing. Exhaustive testing is, of course, infeasible for real-world systems, but some definitions of discrimination, including causal discrimination, are susceptible to provably-sound pruning and adaptive sampling to reduce the test space [28]. Further efficiency improvements and guided test generation are needed to make fairness testing applicable to large, real-world systems. For example, new methods may exploit discovered tests that identify evidence of bias to guide subsequent test generation and may be able to find more evidence faster than random search. Further, fairness testing requires executing the software under test many times, often with similar inputs. This offers an opportunity for improving the efficiency of execution using information from prior executions. This incremental execution optimization could, potentially, greatly reduce test execution time and increase the applicability of fairness testing to larger systems. Similarly, test prioritization and selection may improve the efficiency of fairness testing systems.

As with software quality, while testing can identify fairness bugs, developers also need debugging tools to understand and remove the root causes of the bias. Root causes can lie in the involved requirements, algorithms, implementation, or data [7], and an extensive suite of debugging tools is necessary to support the developers, including tools for debugging how data inaccuracies may affect software outcomes. As an example of data imperfections being a root cause of bias, such imperfections may affect certain sensitive groups more than others: Due to database errors, which are more prevalent in DHS records than SSA records, the E-Verify program (the voluntary, government-run system that employers can use to check whether new employees are work-eligible) had rejection rates 30 times higher for naturalized citizens and 50 times higher for

legal nonimmigrants, than for natural-born citizens [59]. Ideas from research on inferring causal models [44] and relationships [48, 50] in data can be adapted to build models of causal relationships in software behavior, aiding system understanding and debugging tasks. Such fairness analysis has the potential to produce powerful tools because it combines the ability to obtain new execution data on demand by running the software under analysis, and to conduct causal experiments to test causal hypotheses [28].

2.4 Verification

As with all software correctness, verification is a highly desirable goal for software fairness. Recent work proposing verification of fairness properties highlights relevant challenges [2]. First, fairness properties are often aggregates of many executions, as opposed to correctness properties that can be invalidated by a single counterexample execution. This can complicate model-checking and even make checking assertions more challenging. Second, fairness properties are often not only reflective of the software but also of the input population relevant to the software execution, making it more difficult, or less relevant, to verify the software for all populations. Third, fairness properties are often probabilistic, restricting which existing verification techniques can be applied directly to the problem.

Another research challenge with respect to formal verification of fairness properties is identifying ways to encode fairness definitions as verifiable program properties. Some such definitions can be encoded as probabilistic program properties [2], though others may require different mechanisms. Developing runtime environment support for asserting these properties during execution can also help ensure proper behavior.

Finally, support for debugging fairness bugs when formal verification finds behavioral counterexamples or evidence of bias is needed to help developers not only detect bias but also remove it and improve systems.

3 RELATED WORK

Discrimination shows up in many software applications, e.g., advertisements [64], hotel bookings [45], and image search [36]. At the same time, software is entering domains in which discrimination could result in serious negative consequences, including criminal justice [4], finance [56], and hiring [60]. Software discrimination may occur unintentionally, e.g., as a result of implementation bugs, as an unintended property of self-organizing systems [9, 11, 13, 14], as an emergent property of component interaction [10, 12, 15, 41], or as an automatically learned property from biased data [16, 17, 31–34, 73–75].

Existing work on test generation focuses on two specific measures of discrimination, group and causal. Group discrimination is a generalization of the Calders-Verwer (CV) score [17], used frequently in prior work on algorithmic fairness, particularly in the context of fair machine learning [16, 32, 73, 75]. Many other definitions exist [55]. One defines discrimination by observing that a "better" input is never deprived of the "better" output [22]. That definition requires a domain expert to create a distance function for comparing inputs. Causal discrimination [28] measures causality [58]. Fairness in machine learning research is also concerned with causal measures of discrimination, e.g., counterfactual fairness [42]. FairML [1] uses orthogonal projection to co-perturb characteristics, which can mask some discrimination, but find discrimination that is more likely to be observed in real-world scenarios.

FairTest [68] uses manually written tests to measure four kinds of discrimination scores: the CV score and a related ratio, mutual information, Pearson correlation, and a regression between the output and sensitive inputs.

Reducing discrimination in machine learning classifiers [16, 32, 67, 73, 75] is important work that is complementary to the effort the software engineering community needs to put in to create tools and other support for developing fair software.

Some fairness properties can be encoded as probabilistic program properties over a distribution of executions and asserted at runtime or even formally verified with probabilistic guarantees [2]. This work can form the basis of future research on formally verifying fairness properties.

Combinatorial testing may be relevant to fairness testing as it aims to minimize the number of tests needed to explore certain combinations of input characteristics. For example, all-pairs testing generates tests that evaluate every possible value combination for every pair of input characteristics, which can be particularly helpful when testing software product lines [5, 35, 37, 38]. The number of tests needed to evaluate every possible value pair can be significantly smaller than the exhaustive testing alternative since each test can simultaneously contribute to multiple value pairs [20, 35, 69]. Such combinatorial testing optimizations may aid fairness testing. Advances in combinatorial testing, e.g., using static or dynamic analyses for vacuity testing [6, 29] or to identify configuration options that cannot affect a test's output [39], can directly improve efficiency of discrimination testing by identifying that changing a particular input characteristic cannot affect a particular test's output, and thus no causal discrimination is possible with respect to that particular input.

It is possible to detect discrimination in software without explicit access to it. For example, AdFisher [21] collects information on how changes in Google ad settings and prior visited webpages affect the ads Google serves. AdFisher computes a variant of group discrimination, but could be adapted to measure other fairness requirements as well.

Causal relationships in data management systems [26, 46, 47] can help explain query results [51] and debug errors [70–72] by tracking and using data provenance [49]. For software systems that use data management, such provenance-based reasoning may aid testing for causal relationships between input attributes and outputs. Our prior work on testing software that relies on data management systems has focused on data errors [53, 54], whereas this work focuses on testing fairness.

Automated test generation tools, e.g., Randoop [57] and EvoSuite [25] target improving software quality, but their underlying mechanisms may be helpful for generating fairness test suites as well. Fairness testing, however, involves more than generating test suites. It also processes the results of these tests to compute bias. Some generation can use results of the analyses in an iterative loop to determine when to generate more tests and to guide which tests to generate. The goals of traditional test generation tools also differ

from the goal of fairness testing, aiming instead to generate tests for testing goals, e.g., maximizing coverage, which leads to diverse test suites [25, 57]. Fairness testing often needs to generate pairs of similar, not diverse, inputs that traditional tools are unlikely to generate.

4 SUMMARY

Software fairness is an increasingly important problem for today's software that should be tackled from multiple directions. In particular, ensuring software fairness is a software engineering problem, with deep apparent parallels between software fairness and software quality. Requirements elicitation, design, testing, and verification — all critical parts of the software engineering lifecycle and of ensuring software quality — are all relevant and necessary for ensuring software fairness, but significant research challenges exist in creating developer support in each of these areas.

There are early encouraging signs that the software engineering community is getting involved in software fairness, such as the 2018 IEEE/ACM International Workshop on Software Fairness (FairWare 2018). Combined with the machine learning community's efforts, this challenging problem provides numerous opportunities for interdisciplinary, collaborative solutions. The potential for significant societal impact and many challenging and interesting research problems, such as fairness requirements specification, automated test generation, and efficient test selection and execution make these exciting research problems worthy of the attention of software engineering researchers, as well as researchers in data science, machine learning, theoretical computer science, and other disciplines.

ACKNOWLEDGMENT

This work is supported by the National Science Foundation under grants no. CCF-1453474, IIS-1453543, and CNS-1744471.

REFERENCES

[1] Julius Adebayo and Lalana Kagal. Iterative orthogonal feature projection for diagnosing bias in black-box models. *CoRR*, abs/1611.04967, 2016.
[2] Aws Albarghouthi, Loris D'Antoni, Samuel Drews, and Aditya Nori. FairSquare: Probabilistic verification for program fairness. In *ACM International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA)*, 2017.
[3] Paul Ammann and Jeff Offutt. *Introduction to Software Testing*. Cambridge University Press, 1 edition, 2008.
[4] Rico Angell, Brittany Johnson, Yuriy Brun, and Alexandra Meliou. Themis: Automatically testing software for discrimination. In *Proceedings of the Demonstrations Track at the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, Lake Buena Vista, FL, USA, November 2018.
[5] Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. Machine bias. *ProPublica*, May 23, 2016. https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing.
[6] Sven Apel, Alexander von Rhein, Philipp Wendler, Armin Größlinger, and Dirk Beyer. Strategies for product-line verification: Case studies and experiments. In *International Conference on Software Engineering (ICSE)*, pages 482–491, 2013.
[7] Thomas Ball and Orna Kupferman. Vacuity in testing. In *International Conference on Tests and Proofs (TAP)*, pages 4–17, 2008.
[8] Solon Barocas and Andrew D. Selbst. Big data's disparate impact. *California Law Review*, 104, 2016.
[9] Su Lin Blodgett and Brendan O'Connor. Racial disparity in natural language processing: A case study of social media african-american english. In *Fairness, Accountability, and Transparency in Machine Learning (FAT/ML)*, 2017.
[10] Yuriy Brun, Ron Desmarais, Kurt Geihs, Marin Litoiu, Antonia Lopes, Mary Shaw, and Mike Smit. A design space for adaptive systems. In Rogério de Lemos, Holger Giese, Hausi A. Müller, and Mary Shaw, editors, *Software Engineering for Self-Adaptive Systems II*, volume 7475, pages 33–50. Springer-Verlag, 2013.
[11] Yuriy Brun, George Edwards, Jae young Bang, and Nenad Medvidovic. Smart redundancy for distributed computation. In *International Conference on Distributed Computing Systems (ICDCS)*, pages 665–676, Minneapolis, MN, USA, June 2011.
[12] Yuriy Brun and Nenad Medvidovic. An architectural style for solving computationally intensive problems on large networks. In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, Minneapolis, MN, USA, May 2007.
[13] Yuriy Brun and Nenad Medvidovic. Fault and adversary tolerance as an emergent property of distributed systems' software architectures. In *International Workshop on Engineering Fault Tolerant Systems (EFTS)*, pages 38–43, Dubrovnik, Croatia, September 2007.
[14] Yuriy Brun and Nenad Medvidovic. Keeping data private while computing in the cloud. In *International Conference on Cloud Computing (CLOUD)*, pages 285–294, Honolulu, HI, USA, June 2012.
[15] Yuriy Brun and Nenad Medvidovic. Entrusting private computation and data to untrusted networks. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 10(4):225–238, July/August 2013.
[16] Yuriy Brun, Jae young Bang, George Edwards, and Nenad Medvidovic. Self-adapting reliability in distributed software systems. *IEEE Transactions on Software Engineering (TSE)*, 41(8):764–780, August 2015.
[17] Toon Calders, Faisal Kamiran, and Mykola Pechenizkiy. Building classifiers with independency constraints. In *Proceedings of the 2009 IEEE International Conference on Data Mining (ICDM) Workshops*, pages 13–18, 2009.
[18] Toon Calders and Sicco Verwer. Three naive Bayes approaches for discrimination-free classification. *Data Mining and Knowledge Discovery*, 21(2):277–292, 2010.
[19] Aylin Caliskan, Joanna J. Bryson, and Arvind Narayanan. Semantics derived automatically from language corpora contain human-like biases. *Science*, 356(6334):183–186, 2017.
[20] Betty Cheng et al. Software engineering for self-adaptive systems: A research roadmap. In *Software Engineering for Self-Adaptive Systems*, volume 5525, pages 1–26. Springer-Verlag, 2009.
[21] Marcelo d'Amorim, Steven Lauterburg, and Darko Marinov. Delta execution for efficient state-space exploration of object-oriented programs. In *International Symposium on Software Testing and Analysis (ISSTA)*, pages 50–60, 2007.
[22] Amit Datta, Michael Carl Tschantz, and Anupam Datta. Automated experiments on ad privacy settings. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 1:92–112, 2015.
[23] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. Fairness through awareness. In *Innovations in Theoretical Computer Science Conference (ITCS)*, pages 214–226, 2012.
[24] Naeem Esfahani, Sam Malek, and Kaveh Razavi. GuideArch: Guiding the exploration of architectural solution space under uncertainty. In *IEEE/ACM International Conference on Software Engineering (ICSE)*, pages 43–52, 2013.
[25] Anthony Finkelstein, Mark Harman, S. Afshin Mansouri, Jian Ren, and Yuanyuan Zhang. "Fairness analysis" in requirements assignments. In *IEEE International Requirements Engineering Conference (RE)*, pages 115–124, Washington, DC, USA, 2008.
[26] Gordon Fraser and Andrea Arcuri. Whole test suite generation. *IEEE Transactions on Software Engineering (TSE)*, 39(2):276–291, 2013.
[27] Cibele Freire, Wolfgang Gatterbauer, Neil Immerman, and Alexandra Meliou. A characterization of the complexity of resilience and responsibility for self-join-free conjunctive queries. *Proceedings of the VLDB Endowment (PVLDB)*, 9(3):180–191, 2015.
[28] Sorelle A. Friedler, Carlos Scheidegger, and Suresh Venkatasubramanian. On the (im)possibility of fairness. *CoRR*, abs/1609.07236, 2016.
[29] Sainyam Galhotra, Yuriy Brun, and Alexandra Meliou. Fairness testing: Testing software for discrimination. In *European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 498–510, 2017.
[30] Luigi Di Guglielmo, Franco Fummi, and Graziano Pravadelli. Vacuity analysis for property qualification by mutation of checkers. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 478–483, 2010.
[31] Devindra Haweawar. Staples, Home Depot, and other online stores change prices based on your location. *VentureBeat*, December 24, 2012. https://venturebeat.com/2012/12/24/staples-online-stores-price-changes.
[32] Faisal Kamiran and Toon Calders. Classifying without discriminating. In *International Conference on Computer, Control, and Communication (IC4)*, pages 1–6, 2009.
[33] Faisal Kamiran, Toon Calders, and Mykola Pechenizkiy. Discrimination aware decision tree learning. In *International Conference on Data Mining (ICDM)*, 2010.
[34] Faisal Kamiran, Asim Karim, and Xiangliang Zhang. Decision theory for discrimination-aware classification. In *International Conference on Data Mining (ICDM)*, pages 924–929, 2012.
[35] Toshihiro Kamishima, Shotaro Akaho, Hideki Asoh, and Jun Sakuma. Fairness-aware classifier with prejudice remover regularizer. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, pages 35–50, 2012.
[36] Christian Kästner, Alexander von Rhein, Sebastian Erdweg, Jonas Pusch, Sven Apel, Tillmann Rendel, and Klaus Ostermann. Toward variability-aware testing. In *International Workshop on Feature-Oriented Software Development (FOSD)*, pages 1–8, 2012.

[37] Matthew Kay, Cynthia Matuszek, and Sean A. Munson. Unequal representation and gender stereotypes in image search results for occupations. In *Conference on Human Factors in Computing Systems (CHI)*, pages 3819–3828, 2015.

[38] Chang Hwan Peter Kim, Don S. Batory, and Sarfraz Khurshid. Reducing combinatorics in testing product lines. In *International Conference on Aspect-Oriented Software Development (AOST)*, pages 57–68, 2011.

[39] Chang Hwan Peter Kim, Sarfraz Khurshid, and Don Batory. Shared execution for efficiently testing product lines. In *International Symposium on Software Reliability Engineering (ISSRE)*, pages 221–230, 2012.

[40] Chang Hwan Peter Kim, Darko Marinov, Sarfraz Khurshid, Don Batory, Sabrina Souto, Paulo Barros, and Marcelo D'Amorim. SPLat: Lightweight dynamic analysis for reducing combinatorics in testing configurable systems. In *European Software Engineering Conference and ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE)*, pages 257–267, 2013.

[41] Brendan F. Klare, Mark J. Burge, Joshua C. Klontz, Richard W. Vorder Bruegge, and Anil K. Jain. Face recognition performance: Role of demographic information. *IEEE Transactions on Information Forensics and Security (TIFS)*, 7(6):1789–1801, 2012.

[42] Ivo Krka, Yuriy Brun, George Edwards, and Nenad Medvidovic. Synthesizing partial component-level behavior models from system specifications. In *European Software Engineering Conference and ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE)*, pages 305–314, Amsterdam, The Netherlands, August 2009.

[43] Matt J. Kusner, Joshua R. Loftus, Chris Russell, and Ricardo Silva. Counterfactual fairness. In *Annual Conference on Neural Information Processing Systems (NIPS)*, 2017.

[44] Rafi Letzter. Amazon just showed us that 'unbiased' algorithms can be inadvertently racist. *TECH Insider*, April 21, 2016. http://www.techinsider.io/how-algorithms-can-be-racist-2016-4.

[45] Marc Maier, Katerina Marazopoulou, David Arbour, and David Jensen. A sound and complete algorithm for learning causal models from relational data. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 371–380, 2013.

[46] Dana Mattioli. On Orbitz, Mac users steered to pricier hotels. *The Wall Street Journal*, August 23, 2012. http://www.wsj.com/articles/SB10001424052702304458604577488822667325882.

[47] Alexandra Meliou, Wolfgang Gatterbauer, Joseph Y. Halpern, Christoph Koch, Katherine F. Moore, and Dan Suciu. Causality in databases. *IEEE Data Engineering Bulletin*, 33(3):59–67, 2010.

[48] Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu. The complexity of causality and responsibility for query answers and non-answers. *Proceedings of the VLDB Endowment (PVLDB)*, 4(1):34–45, 2010.

[49] Alexandra Meliou, Wolfgang Gatterbauer, and Dan Suciu. Bringing provenance to its full potential using causal reasoning. In *USENIX Workshop on the Theory and Practice of Provenance (TaPP)*, 2011.

[50] Alexandra Meliou, Wolfgang Gatterbauer, and Dan Suciu. Bringing provenance to its full potential using causal reasoning. In *3rd USENIX Workshop on the Theory and Practice of Provenance (TaPP)*, 2011.

[51] Alexandra Meliou, Wolfgang Gatterbauer, and Dan Suciu. Reverse data management. *Proceedings of the VLDB Endowment (PVLDB)*, 4(11):1490–1493, 2011.

[52] Alexandra Meliou, Sudeepa Roy, and Dan Suciu. Causality and explanations in databases. *Proceedings of the VLDB Endowment (PVLDB) tutorial*, 7(13):1715–1716, 2014.

[53] Jakub Mikians, László Gyarmati, Vijay Erramilli, and Nikolaos Laoutaris. Detecting price and search discrimination on the Internet. In *ACM Workshop on Hot Topics in Networks (HotNets)*, pages 79–84, 2012.

[54] Kıvanç Muşlu, Yuriy Brun, and Alexandra Meliou. Data debugging with continuous testing. In *European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) NIER track*, pages 631–634, 2013.

[55] Kıvanç Muşlu, Yuriy Brun, and Alexandra Meliou. Preventing data errors with continuous testing. In *ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, pages 373–384, 2015.

[56] Arvind Narayanan. 21 fairness definitions and their politics. Tutorial at the Conference on Fairness, Accountability, and Transparency, 2018.

[57] Parmy Olson. The algorithm that beats your bank manager. *CNN Money*, March 15, 2011. http://www.forbes.com/sites/parmyolson/2011/03/15/the-algorithm-that-beats-your-bank-manager/#cd84e4f77ca8.

[58] Carlos Pacheco, Shuvendu K. Lahiri, Michael D. Ernst, and Thomas Ball. Feedback-directed random test generation. In *ACM/IEEE International Conference on Software Engineering (ICSE)*, pages 75–84, 2007.

[59] Judea Pearl. Causal inference in statistics: An overview. *Statistics Surveys*, 3:96–146, 2009.

[60] Marc R. Rosenblum. E-Verify: Strengths, weaknesses, and proposals for reform, 2011.

[61] Aarti Shahani. Now algorithms are deciding whom to hire, based on voice. *NPR All Things Considered*, 2015.

[62] Arman Shahbazian, Youn Kyu Lee, Yuriy Brun, and Nenad Medvidovic. Poster: Making well-informed software design decisions. In *International Conference on Software Engineering (ICSE) Poster*, pages 262–263, 2018.

[63] Arman Shahbazian, Youn Kyu Lee, Duc Le, Yuriy Brun, and Nenad Medvidovic. Recovering architectural design decisions. In *IEEE International Conference on Software Architecture (ICSA)*, pages 95–104, 2018.

[64] Seung Yeob Shin, Yuriy Brun, Hari Balasubramanian, Philip Henneman, and Leon Osterweil. Discrete-event simulation and integer linear programming for constraint-aware resource scheduling. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2018.

[65] Latanya Sweeney. Discrimination in online ad delivery. *Communications of the ACM (CACM)*, 56(5):44–54, 2013.

[66] Rachael Tatman. Gender and dialect bias in YouTube's automatic captions. In *Workshop on Ethics in Natural Language Processing*, 2017.

[67] Richard N. Taylor, Nenad Medvidovic, and Eric M. Dashofy. *Software Architecture: Foundations, Theory, and Practice.* John Wiley & Sons, 2009.

[68] Philip S. Thomas, Bruno Castro da Silva, Andrew G. Barto, and Emma Brunskill. On ensuring that intelligent machines are well-behaved. *CoRR*, abs/1708.05448, 2017.

[69] Florian Tramer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, Jean-Pierre Hubaux, Mathias Humbert, Ari Juels, and Huang Lin. FairTest: Discovering unwarranted associations in data-driven applications. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2017.

[70] Alexander von Rhein, Sven Apel, and Franco Raimondi. Introducing binary decision diagrams in the explicit-state verification of Java code. In *The Java Pathfinder Workshop (JPF)*, 2011.

[71] Xiaolan Wang, Xin Luna Dong, and Alexandra Meliou. Data X-Ray: A diagnostic tool for data errors. In *International Conference on Management of Data (SIGMOD)*, 2015.

[72] Xiaolan Wang, Alexandra Meliou, and Eugene Wu. QFix: Demonstrating error diagnosis in query histories. In *International Conference on Management of Data (SIGMOD) demonstration track*, pages 2177–2180, 2016.

[73] Xiaolan Wang, Alexandra Meliou, and Eugene Wu. QFix: Diagnosing errors through query histories. In *International Conference on Management of Data (SIGMOD)*, 2017.

[74] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez Rodriguez, and Krishna P Gummadi. Learning fair classifiers. *CoRR*, abs/1507.05259, 2015.

[75] Richard Zemel, Yu (Ledell) Wu, Kevin Swersky, Toniann Pitassi, and Cynthia Dwork. Learning fair representations. In *International Conference on Machine Learning (ICML), published in JMLR W&CP: 28(3):325–333)*, 2013.

[76] Indre Žliobaite, Faisal Kamiran, and Toon Calders. Handling conditional discrimination. In *International Conference on Data Mining (ICDM)*, 2011.