Ground-Truth Deficiencies in Software Engineering: When Codifying the Past Can Be Counterproductive

Eray Tüzün, Bilkent University

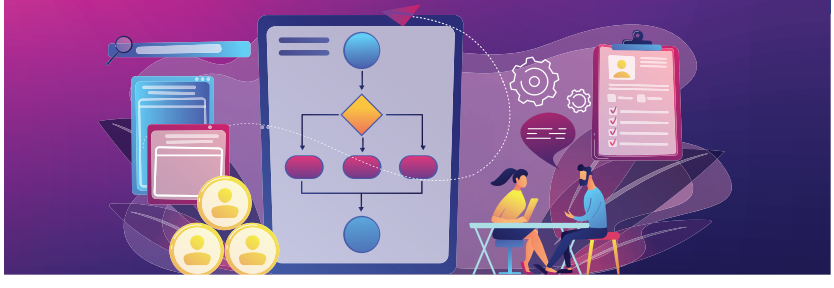
Hakan Erdogmus, Carnegie Mellon University

Maria Teresa Baldassarre, University of Bari

Michael Felderer, University of Innsbruck and Blekinge Institute of Technology

Robert Feldt, Chalmers University of Technology and Blekinge Institute of Technology

Burak Turhan, University of Oulu and Monash University



Digital Object Identifier 10.1109/MS.2021.3098670
Date of current version: 18 April 2022

// In software engineering, the objective function of human decision makers might be influenced by many factors. Relying on historical data as the ground truth may give rise to systems that automate software engineering decisions by mimicking past suboptimal behavior. We describe the problem and offer some strategies. //

AUTHOR CATHY O'NEILL[1] issues a stern warning about relying on historical data to automate decision-making processes in her book *Weapons of Math Destruction*. The warning is "Big Data processes codify the past. They do not invent the future." She is referring to the built-in biases present in past data, which get reinforced if those data are used to automate future decisions, leading to an unvirtuous cycle with potentially negative social justice implications.

In his *Harvard Business Review* article, Redman[2] confirms this effect: "Poor data quality is enemy number one to the widespread, profitable use of machine learning. The quality demands of machine learning are steep, and bad data can rear its ugly head twice both in the historical data used to train the predictive model and in the new data used by that model to make future decisions."

Data science and machine learning expertise in software development

teams is becoming more pervasive and central.[3] Software organizations have been taking advantage of teams with combined engineering and data science skills to analyze, improve, and automate their organizations' development processes and decisions based on data collected internally or from external and public sources, with applications ranging from predicting defects[4] to the assignment of engineers to various development tasks.[5] Considering the increasing popularity of these and other similar machine learning applications in software engineering, we make the parallel case that blindly relying on historical data to automate software engineering decisions can be harmful.

The use of rich historical data undoubtedly improved our ability to answer many questions about software engineering practice in novel ways and led to important advances. However, software engineering is not uniquely immune to problems that plague applications in other fields. In software engineering, the harm may not be a deeply ingrained social injustice but is more likely to involve perpetual suboptimality. While seemingly benign, this suboptimality may end up deteriorating rather than improving decision-making quality and, ultimately, compromise the software practices that rely on these decisions.

Like in the general case, the root cause of suboptimality in software engineering can often be traced to various kinds of cognitive biases.[6] Typically, the historical data misrepresents the dependent construct involved in a decision problem—say, choosing a qualified developer for a specific technical task—by knowingly or unknowingly substituting a critically important attribute (e.g., technical competence) by an attribute much less relevant to the decision's goal (e.g., positive social interactions). Typically, the substitution is not just random noise but happens systemically.

From a research perspective, the end result is a possibly serious threat to construct validity. However, research-related considerations, which are well known in empirical software engineering,[7] are not our focus. Instead, we focus on practical implications by addressing the consequences tied to the resulting models' use as black boxes in real tools and, ultimately, in real practice without any concerns about the data with which the models were trained.

Biases become reinforced when we simply codify faulty past behavior in a tool automating a decision problem. Arguably, there may be short-term value in automating even bad decisions because they save time, but long-term damage may erase any immediate savings.

To avoid harming future practice, the dependent construct in a decision-making problem should at least approximate a defensibly good answer. The objectively correct, or optimal, answer is what *ground truth* normally means, but it is sufficient for it to be good enough. If the ground truth is systemically of poor quality, the decisions will also be systemically poor, degrading future practice rather than supporting it. More importantly, the resulting system's poor performance can never be revealed: when the ground truth is distorted severely by suboptimal past outcomes, the reference point to be used in any validation task would, by definition, be wrong.

Although well documented in the general machine learning literature,[8] the practical implications of ground-truth deficiencies are not widely recognized in software engineering. Menzies and Shepperd[9] list a number of "bad smells" in software analytics applications, but they do not explicitly allude to ground-truth issues. *IEEE Software*'s special issue on "50 Years of Software Engineering"[10] discusses several challenges in the same context without bringing up the ground truth. Our goal in this article is to close this gap and raise awareness. To that end, we first discuss cases drawn from familiar software engineering applications. Building on these cases, we propose a structured discovery and improvement process.

This article builds on expert insights gleaned from a working session conducted on the topic at the 2019 annual meeting of the International Software Engineering Research Network (ISERN 2019). The session is described in "Investigating the Validity of the Ground Truth in Software Engineering at ISERN 2019." Many of the cases presented originated from that context.

Motivating Example

As a first example, consider the modern code review process, where the goal is to select qualified reviewers for a new pull request (PR). Many code reviewer recommendation (CRR) systems train their models on data sets gathered from real-life projects, using actual past assignments to represent the ground truth.

We can think of the code reviewer decision as a classification problem. Each data point corresponds to a PR with the assigned code reviewer being the label assigned by a human. The top row of Table 1 shows one example that builds a Bayesian network from past assignments.

In CRR systems, the code reviewer assigned to a past PR is assumed to be qualified to perform the review. However, in many situations, the assigned code reviewer may not necessarily be a

good choice, or the good choices may not even be represented in the data set.

In reality, the objective function of a human decision maker for recommending a reviewer might be implicit (e.g., based on convenience and subject to availability, recency, recall, default, and wishful-thinking biases[6]) and fail to align with the ideal objective function required for an optimally performing system (e.g., based on the reviewer's technical competence and familiarity with the piece of code involved in the PR). Thus, the ground truth may be distorted by cognitive biases affecting the human decision makers who make the assignments. When this happens, the resulting systems will, at best, mimic poorly made past decisions.

When Ground Truth Is Objective

To give a counterexample in which there is no apparent ground-truth distortion of the kind discussed, consider the work by Owhadi-Kareshk et al,[17] who attempt to predict merge conflicts where a file in a codebase is modified simultaneously by multiple developers. The purpose of the prediction is to increase the efficiency of the code integration process by alerting the involved developers of impending conflicts in a timely way.

This would aid in speculative merging by eliminating expensive real-time checks: file change combinations that are unlikely to lead to merge conflicts would not need these checks. The ground truth is represented by the actual merge conflicts in the historical data, which can be objectively determined. Unlike the CRR scenario described, where labelers were humans subject to biases, an algorithm performed the labeling in the merge conflict data set with 100% accuracy.

Applications With Potential Ground-Truth Deficiencies

Table 1 illustrates the pervasiveness of ground-truth problems through additional representative cases. The cases constitute a convenience sample (see "Investigating the Validity of the Ground Truth in Software Engineering at ISERN 2019"). Each case presents a well-motivated application and uses the proper methods but suffers from potential ground-truth issues worthy of explicating and checking. Some of these issues have been acknowledged and addressed by the original authors to varying extents, while others remain outstanding. We use the cases' contexts as illustrative examples to raise awareness and propose concrete strategies.

Next, let's focus on another familiar application, defect prediction (DP). In the example case,[14] the authors use cross-project data from

INVESTIGATING THE VALIDITY OF THE GROUND TRUTH IN SOFTWARE ENGINEERING AT ISERN 2019

This article culminated from the session on "Investigating the Validity of Ground Truth in Software Engineering" that was held during ISERN 2019, the annual meeting of the ISERN, on 2019 September 17 at Porto de Galinhas, Brazil. A total of 42 expert participants discussed the implications of ground-truth problems in software practice, motivated by the expectation that such problems would become increasingly pervasive with the easy availability of rich historical data, proliferation of machine learning techniques that leverage such data, and gradual penetration of resulting black box models into automated tools that support software development decisions in the field.

During the working session, 10 breakout groups, based on their collective knowledge and experience, were asked to produce structured cases with a potential to negatively impact decision making in software engineering. The results were presented, tabulated, and organized in a format similar to that of Table 1. The examples discussed in this article, including the cases presented in the table, are sourced from the working session results, which were validated with the participants offline following the session by soliciting feedback on the observations and examples recorded.

We thank all participants for their contributions. More information on the ISERN 2019 meeting can be found at http://eseiw2019.com/isern/.

Table 1. Potential ground-truth problems in example applications.

Application and example	Approach	Ground truth	Possible assumption violations
CRR Recommend the best code reviewers for a new PR Jeong et al. (2009)[11]	Use the previous PR reviewer assignment history and commit info (the file location, name of folders, and commit author) to build a Bayesian network using historical data and selected features to predict future reviewers.	Captured by code reviewers who performed an actual review for closed/merged PRs Subject to the following assumptions: • A good reviewer performs the PR efficiently and thoroughly. • Actual past reviewers (whether self-selected or assigned) are always the "best reviewers" for a PR. • A merged/closed PR must have had an effective review.	• Reviewer assignments may be dictated by convenience factors (willingness, workload, or social relationships) rather than technical factors (competence, experience, or familiarity with the code). • A merged/closed PR may cause future bugs and new PRs to fix them.
Reopened bug prediction Predict which bugs will be reopened in an issue repository Shihab et al. 2013[12]	Use the available bug attributes (bug closing time, priority, severity, reporter/fixer of the bug, description, comments, and so on) from the issue repository to build decision-tree-based predictive models using work habits, bug reports, and the bug fixer's characteristics.	Captured by bug reports that were reopened according to historical data Subject to the following assumption: • When a bug resurfaces, a human decision maker correctly identifies the original bug report and reopens it instead of opening a new bug report.	• Bug identifications may be wrong due to recollection problems, turnover causing loss of project memory, convenience factors (it is easier to create a new report than identify the original bug report), optics (reopened bugs may make a developer look bad), and twisted incentives (bug reporters might get more credit for new bugs).
Sentiment analysis Gauge developers' reactions to and contentment with software development technologies Lin et al. (2018)[13]	Use https://stackoverflow.com comments about software tools and libraries or, generally in sentiment analysis, various sources, such as code comments, discussion boards, PR conversations, question-and-answer sites, and commit messages to analyze short strings and predict actual emotion/sentiment based on machine learning models and similarity to known examples (analogy-based methods).	Captured by a post hoc-judged emotion/sentiment a developer had when writing the text. Subject to the following assumptions: • A human or the same developer is able to predict the emotion he or she had when writing the text. • There is a single/dominant emotion/sentiment at a single point in time when the text was written. • Individuals write text that reflects their emotions/sentiments. • People care about being truthful when writing comments online.	• Emotions often cannot be judged post hoc even by the same person writing the text. • Existing models of emotion acknowledge they are complex and interlocking. • Social filters may prevent actual emotions from being accurately expressed in writing.
Defect prediction Predict defect-prone modules in a codebase Turhan et al. (2013)[14]	Use the project data from multiple/mixed projects across different organizations and sources, including code-level metrics, as features and modules found to be defective to build naive Bayes classifiers to predict defect-prone modules based on code-level features from mixed projects.	Captured by defects attributed to modules, both manually and automatically, using an existing heuristic Subject to the following assumptions: • For manual attribution: when a defect is reported for a module, the module is defective, and vice versa. • For automated attribution: the heuristic used is 100% accurate.	• Manual defect reporting/labeling procedures are unclear, might rely on the recollection and accuracy of humans, and may be influenced by political and personal interests. • The automatic defect attribution heuristic may be far less than 100% accurate (an example of an existing prediction method used as the ground truth for another predictor).

(*Continued*)

Table 1. Potential ground-truth problems in example applications. (cont.)

Application and example	Approach	Ground truth	Possible assumption violations
Rework estimation Predict the time it will take to fix a bug Weiss et al. (2017)[15]	Use issue reports (description, severity, associated artifacts, and so on), issue creation, and resolution/closure times to analyze text similarities between new and fixed issues and then average over past bug-fix times for similar issues.	Captured by the time to fix a bug as determined from the issue reporting system Subject to the following assumption: • The time to fix a bug can accurately be computed from timestamp information in the issue reporting system.	• The difference between the bug close and open times can be affected by a variety of factors unrelated to the effort required, such as nonuniform workloads, interruptions, parallel work, and availability. • An issue may have been fixed without having been explicitly closed.
Bug assignment Assign a new bug to a developer for fixing Hu et al. (2014)[16]	Use bug reports to match a new bug to an existing similar bug and associate bugs with components, version-control data to associate components with developers, and real bug–developer assignments for validation to build a weighted graph linking bugs to components and components to developers as well as infer from these relationships a relationship linking bugs to developers that allows a new bug to be assigned to a developer who has fixed a similar bug.	Captured by the bug–component and component–developer relationships, bug–developer assignments, or identity of bug-fix committers Subject to the following assumptions: • For validation purposes: the past bug–developer assignments represent the best possible choices. • The assigned developers are the best choices in each case. • The developer who committed the bug fix actually fixed the bug.	• A bug may not be adequately described by developer-provided information. • A bug may be fixed by one developer and committed by another, changing the assignment. • Bug assignments may be made in a suboptimal way.

multiple sources to predict modules that are likely to contain defects. This job is important for directing limited quality assurance resources to parts of the code that provide the best return on the quality assurance effort. The authors built a Bayesian classifier based on historical data, where the ground truth is represented by attributions of the defects to code artifacts. The attribution, or labeling, was done by humans for part of the data set and automatically using a heuristic in the remainder.

This case is illustrative because it highlights the problems associated with both manual and automated labeling. For models built to be useful in practice, the defect attributions must be reasonably accurate. Manual defect attribution is notorious for being inaccurate; haphazard; and subject to political, convenience, and self-interest biases. Automated defect attribution presents a different kind of problem, where we use one predictive heuristic, whose accuracy may be uncertain, to capture the ground truth for another predictive heuristic.

Table 1 includes other common cases involving prediction tasks that may exhibit similar, potentially harmful ground-truth problems in the historical data: rework estimation (RE) and reopened bug prediction (RBP). In these applications, cognitive biases, such as recall, default, availability, conflict-of-interest, and self-interest effects,[6] could easily be present in, or even dominate, the ground truth.

In the RBP case, the set of reopened bug reports in the historical data is likely to be a subset of the truly recurring bugs. The underrepresentation of bugs may be due to several root-cause biases: not remembering all previous bugs (primacy and recency effects), turnover causing a loss of project memory, not being aware of previous bugs, laziness (it is easier to create a new report than to identify the original bug report), politics (reopened bugs may make a developer look bad), and misguided incentives (bug reporters might get more credit for new bugs).

These biases, if present, will increase false negatives in real practice, missing bugs that may be reopened and preventing early remedial action. In the RE case, biases and inaccuracies in the recorded rework effort data may lead to misguided resource allocation, prioritization, and commitment decisions for development teams, wasting precious resources and causing reputational damage.

In the sentiment analysis (SA) case, ground-truth problems result from inaccuracies by human labelers when attributing emotions to pieces of text. Various cognitive biases—in particular, miserly information, impact, representativeness, selective perception, recency, recall, time based, confirmation, fixation, and invincibility[6]—may contribute to these inaccuracies. Consequently, software technology recommendations from tools and systems that build on models using the sentiment data can become mistrustful and lead to choices that do not meet users' goals.

In the bug assignment (BA) case, the ground truth is represented by three different sources of historical data:

- past bug descriptions
- inferred bug–developer relationships for making recommendations
- actual bug–developer assignments for validating these recommendations.

Although poor bug descriptions will lead to poor recommendations, this example highlights a different issue: performance-masking problems caused by the validation data. Here, the validation data assume past BAs were optimal and unbiased, whereas, in reality, the data may have been tainted by a spectrum of cognitive biases as before, including default, availability, anchoring, adjustment, miserly information, recall, validity, representativeness, fixation, confirmation, conflict of interest, and invincibility.[6]

As a result, the validation data may underestimate the goodness of the approach because the data used to build the recommendation heuristic may, in fact, be more reliable than the data used to validate it. The heuristic might, in practice, perform much better than the validation findings suggest yet be dismissed based on poor performance results. These cases demonstrate how hidden biases that creep into the ground-truth data may defeat the purpose of the applications that depend on them.

A Ground-Truth Improvement Process

The ground truth implies veracity, i.e., what we use as the ground truth in building new models is what is assumed to be true. However, the veracity of the ground-truth cannot simply be viewed as black or white. In practice, what is used as the actual outcomes or labels can be more or less true to the concept we actually want to measure. Since we clearly want high veracity, how can we be aware of ground-truth problems, and what practical guidelines can help us make our ground truth more veritable?

Based on the case examples described, we first worked from the bottom up to group potential ground-truth problems. This allowed a few main clusters to emerge. It was clear that the groups had a natural order, based on either

- the distance from the actual person who should have, ideally, been the source of the ground truth
- the distance from the situation and time in which that person would have been the most equipped to act as that source.

For each major group, we then identified the main types of biases and looked for general mitigation strategies to address them.

Figure 1 shows the results of this iterative aggregation work. The five blue boxes contain prioritized questions that guide the users of historical data toward improving the veracity of their data sets' ground truth. The parallelograms contain high-level summaries of what to do and consider in each step. The top-level question is whether the ground truth involves a human decision maker. However, even in cases when a human decision maker is not involved, it is still important to consider if the labeling is objective and accurate and to demonstrate this.

The main, rightward flow passes through the three essential bias groups we found in our case examples. If the labels were provided by a third party, there could be a multitude of biases due to differences between the third parties and the actual developers and engineers involved. The main biases to consider are related to incomplete and imperfect information of the third parties as well as differences in perception, cognition, views, and emotions.

The next important question to consider is the time between when the ground truth should ideally have been collected and when it was actually recorded. Whenever the ground truth is not recorded in real time, i.e., when the activity the ground-truth data are actually about takes place before the time it was re-enacted and recorded, there is a risk that the data will differ from what they should have been.

Finally, even if the source of the ground truth is not a third party and there is not a significant time difference, any human involved in labeling will still be susceptible to biases. Some of the biases are typically conscious, what we have called *agenda based* in Figure 1. However, others may be subconscious, i.e., based on convenience or individual views and expectations. Even though human factors can affect all of the three steps, conscious factors are directly in focus in the last step,

where more subtle behaviors should be carefully considered.

The process shown in Figure 1 has a natural progression from right to left. Once biases related to the distance between the third party and direct involvers have been considered and improved upon, we can consider biases related to time differences. When these have been accounted for, the process guides us to also consider other biases originating from subjectivity and self interest.

If, at some point, we consider our labeling process so formalized and objective that no further human biases can be considered, the process

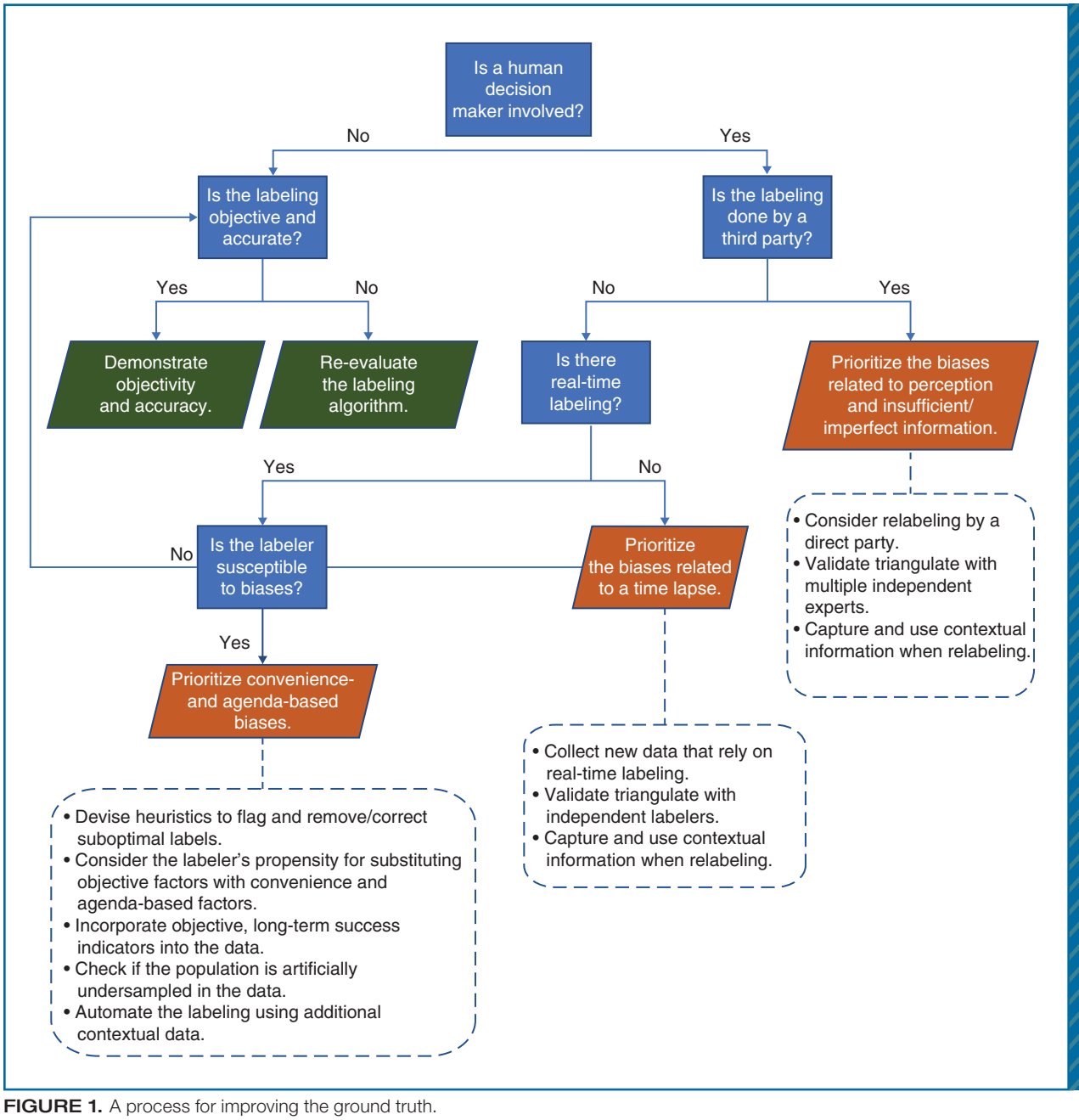


FIGURE 1. A process for improving the ground truth.

encourages us to also consider any potential issues in the labeling process itself. We argue that this is a natural order in which to examine the veracity of the ground-truth data: the more to the right of the process flow we are, the more susceptible we are to underlying biases and the more kinds of biases there are to which we are susceptible.

Improvement Strategies

The process in Figure 1 can be used to examine the quality of the ground-truth data in a systematic way. Given the contextual richness of software engineering applications, it is impossible to cover every plausible scenario. A one-size-fits-all solution does not exist, but we can still focus on general alleviation strategies that fit common recurring contexts.

Figure 1 lists some common strategies in the dashed boxes. This is only our initial attempt; future work should consider more refined guidelines for improving ground-truth veracity in software engineering in different application contexts.

To illustrate the strategies more concretely, consider a situation in which the ground-truth labels are determined by a third party. Assume we have limited access to the people and processes involved in determining the ground truth, which makes it susceptible to a wide range of biases related to perceptions, convenience, self-interest, and imperfect information. This situation corresponds to the rightmost parallelogram in Figure 1.

The ideal remedial strategy to consider in this case would be to relabel all data by systematic and transparent techniques involving objective parties and experts. While this strategy can be impractically expensive after the fact, it has been implemented for applications requiring defect identification and attribution (BA, DP, RBP cases in Table 1).

An example is SmartSHARK,[18] a human-in-the-loop, crowdsourcing approach for labeling defect-related ground-truth data. SmartSHARK uses multiple experts with rigorous contribution and agreement protocols. Notably, the agreement protocols can force labelers to leverage contextual information. The use of contextual information can also help in other situations with high levels of subjectivity, for example, in sentiment identification (the SA case in Table 1).

> Even if the source of the ground truth is not a third party and there is not a significant time difference, any human involved in labeling will still be susceptible to biases.

Next, consider a situation where the data are collected directly but not in real time, corresponding to the middle parallelogram in Figure 1. The ideal strategy would be to switch to the real-time collection of new data. However, this may be infeasible in many cases. For example, processing open source project data is almost impossible with real-time, human-in-the-loop labeling.

The good news is that the two strategies mentioned in the first situation—after-the-fact validation by multiple, independent experts and the use of additional contextual information to triangulate and correct the ground-truth data—apply here as well. Examples of these techniques can be found in Garcia et al,[19] where architectural ground truths recovered from various artifacts (code and documents) and in multiple ways were certified by a software architect with firsthand knowledge about these artifacts.

Suppose we have solved third-party labeling and time-separation issues, and we find ourselves in a situation in which directly involved humans have captured the ground truth in near real time. This corresponds to the parallelogram in the bottom left corner of Figure 1. Using multiple experts in real time is often impractical, and we would have to rely on a single labeler who may be susceptible to biases.

Thus, instead, we could resort to automated labeling heuristics and triangulating/validating with additional data sources. The labeling heuristic can rely on secondary information collected through instrumentation. For example, in the RE application of Table 1, a bug-fix effort can be inferred from or validated by fine-grained telemetry data on relevant developer actions as well as additional sources containing information on idle times, meetings, and workloads.

Another application where triangulation and secondary information can be useful is SA. In the SA case of Table 1, declared first-person sentiments can be validated through biometric measurements from wearable tech (secondary information). In addition,

if text analysis is used to infer sentiments, experts can revisit the labels to correct possible misclassifications.

In the CRR case of Table 1, such after-the-fact correction may be possible by tracing the long-term effects of a reviewer assignment to a PR—e.g., by linking future bugs to past PRs via the bug-tracking system—and removing/correcting the review assignments for these potentially unsuccessful PRs in the historical data. A similar approach may also be implemented in the BA case: a success measure can be defined using available contextual information to more objectively evaluate whether a bug-fix assignment was successful, e.g., by ensuring that the bug that was subject to an assignment was, in fact, never reopened again.

The cases presented are not collectively exhaustive: some applications may require custom strategies. For example, we may need a buffer period in data collection to allow for latent effects to be recorded when such effects exist. A buffer period may be warranted in the DP application to include latent bugs from recent releases in the ground-truth data.

When ground-truth labels are highly subjective, training the labelers may make sense. However, this approach is not without caveats: while training may provide short-term benefits, it may be too expensive without clear long-term returns.[6]

If the original labelers are accessible, another human-in-the-loop validation strategy is to revisit the data with them to identify suboptimal labels and reveal biases. Asking directed questions on how the decisions and estimates were reached and challenging them in a retrospective session might provide cues on how reliable the data are, raise awareness, and help improve the reliability of future data originating from the same labelers.

Validating and correcting ground-truth data after the fact is important, but, even with best efforts, it may still not be enough. Ultimately, we may need to collect new data with better labeling processes.

> Cleaning up the ground-truth data by removing suspect samples improved the performance of the evaluated CRR techniques.

Relying blindly on historical data as the ground truth may give rise to automated solutions that end up mimicking past suboptimal behavior. Reconstructing ground-truth post facto is especially susceptible to several biases. We compiled a list of typical cases that illustrate the pervasiveness of ground-truth problems in software engineering and provided a prioritization and remediation process. The central features of this process are as follows:

- If ground-truth data involve a human decision maker, the priority should be to collect the relevant data directly—that is, in real time and without relying on a third party.
- When this is not feasible, revisit the assumptions made about the ground-truth data and identify possible violations. The majority of the assumption violations would stem from cognitive biases, which may need to be addressed. Table 1 gives typical examples.
- Once the assumption violations are exposed, use the process given in Figure 1 as a guide to improve the ground-truth quality. If possible, validate and correct the data using secondary sources. If not, consider collecting new data using better strategies. If none of this is feasible, focus efforts on improving future data collection.

In recent work, we have applied the ground-truth improvement process and some of the strategies to our motivating example, the CRR problem.[20] In that work, the labeling was done in real time by parties directly involved with reviewer assignments, traversing the blue rectangles in Figure 1 from the right- to leftmost ("Is the labeling done by a third party?" "Yes." "Is there real-time labeling?" "Yes." "Is the labeler susceptible to biases?" "Yes").

After recognizing that the original labelers could have been prone to convenience biases, we looked for more objective, longer-term success factors in the data that confirm or refute the labeling decisions and devised a heuristic to flag and remove the samples that violated the identified success factor (thus following the strategies mentioned in the bottom leftmost dashed box). Cleaning up the ground-truth data by removing suspect samples improved the performance of the evaluated CRR techniques. This application, however, is just one case, and we need many more cases that

ABOUT THE AUTHORS

ERAY TÜZÜN is an assistant professor leading the Software Engineering and Data Analytics Research Group in the Department of Computer Engineering, Bilkent University, Ankara, 06800, Turkey. His research interests include software analytics, empirical software engineering, and software reuse. Tüzün received his Ph.D. from Middle Eastern Technical University, Turkey. He is a Senior Member of IEEE. Contact him at eraytuzun@cs.bilkent.edu.tr.

MICHAEL FELDERER is an associate professor in the Department of Computer Science, University of Innsbruck, Innsbruck, 6020, Austria, and a guest professor in the Department of Software Engineering at the Blekinge Institute of Technology, Sweden. His research interests include software quality and testing, artificial intelligence and software engineering, and requirements engineering. Felderer received his habilitation degree from the University of Innsbruck. Contact him at michael.felderer@uibk.ac.at.

HAKAN ERDOGMUS is a teaching professor in the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania, 15213, USA. His research interests include empirical software engineering, software quality, and software economics. Erdogmus received his Ph.D. from the University of Québec, Montréal. He is a Gold Core member of the IEEE Computer Society and a Senior Member of IEEE. Contact him at hakan.erdogmus@sv.cmu.edu.

ROBERT FELDT is a professor of software engineering at Chalmers University of Technology, Gothenburg, SE-412 96, Sweden, and Blekinge Institute of Technology, Sweden. His research interests include human factors, automation and statistics, and software testing and quality. Feldt received his Ph.D. in computer engineering from the Chalmers University of Technology. He is coeditor in chief of *Empirical Software Engineering* and on the editorial board of two other journals. Contact him at robert.feldt@chalmers.se.

MARIA TERESA BALDASSARRE is an associate professor at the Department of Informatics, University of Bari, Bari, Italy. Her research interests include empirical software engineering, human factors in software engineering, and software measurement. Baldassare received her Ph.D. from the University of Bari. She is the representative of the University of Bari in the International Software Engineering Research Network and an associate editor of *Decision Support Systems*. Contact her at mariateresa.baldassarre@uniba.it.

BURAK TURHAN is a professor of software engineering at the University of Oulu, Oulu, FI-90014, Finland, and an adjunct professor (research) in the Faculty of Information Technology at Monash University. His research interests include empirical software engineering, software analytics, and quality assurance and testing. Turhan received his Ph.D. from Boğaziçi University. He is a Senior Member of IEEE and the Association for Computing Machinery. Contact him at turhanb@computer.org.

traverse Figure 1 in different ways, activating different improvement strategies, to validate the advice.

The ground truth consists of shades of gray. Even if we apply these steps, we cannot entirely eliminate all problems. The goal is to make sure that the resulting systems are not perpetually bound to problematic data

but have a chance to improve over time with new and better data and data collection processes.

In situations where biases and suboptimal behavior are pervasive and impossible to detect, past data can only end up capturing seriously flawed practice. Classical optimization approaches may be preferable to a data-centered approach in these situations.

Acknowledgments

Eray Tüzün and Hakan Erdogmus contributed equally to this work as first authors. The remaining authors are listed in alphabetical order.

References

1. C. O'Neil, *Weapons of Math Destruction: How Big Data Increases Inequality and Threatens Democracy*. New York: Crown Publishing Group, 2016.
2. T. C. Redman, "If your data is bad, your machine learning tools are useless," Harvard Business Review, Apr. 2, 2018. https://hbr.org/2018/04/if-your-data-is-bad-your-machine-learning-tools-are-useless (accessed Mar. 1, 2021).
3. M. Kim, T. Zimmermann, R. DeLine, and A. Begel, "The emerging role of data scientists on software development teams," in *Proc. 38th Int. Conf. Softw. Eng., (ICSE '16)*, New York: Association for Computing Machinery, 2016, pp. 96–107. doi: 10.1145/2884781.2884783.
4. D. Bowes, S. Counsell, T. Hall, J. Petric, and T. Shippey, "Getting defect prediction into industrial practice: The ELFF tool," in *Proc. 2017 IEEE Int. Symp. Softw. Rel. Eng. Workshops (ISSREW)*, pp. 44–47. doi: 10.1109/ISSREW.2017.11.
5. L. Jonsson, M. Borg, D. Broman, K. Sandahl, S. Eldh, and P. Runeson, "Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts," *Empirical Softw. Eng.*, vol. 21, no. 4, pp. 1533–1578, Aug. 2016. doi: 10.1007/s10664-015-9401-9.
6. R. Mohanani, I. Salman, B. Turhan, P. Rodríguez, and P. Ralph, "Cognitive biases in software engineering: A systematic mapping study," *IEEE Trans. Softw. Eng.*, vol. 46, no. 12, pp. 1318–1339, Oct. 2018. doi: 10.1109/TSE.2018.2877759.
7. R. Feldt and A. Magazinius, "Validity threats in empirical software engineering research-an initial survey," in *Proc. 22nd Int. Conf. Softw. Eng. Knowl. Eng. (SEKE)*, 2012, pp. 374–379.
8. H. Jiang and O. Nachum, "Identifying and correcting label bias in machine learning," in *Proc. Mach. Learn. Res.*, 2020, vol. 108, pp. 702–712.
9. T. Menzies and M. Shepperd, "'Bad smells' in software analytics papers," *Inf. Softw. Technol.*, vol. 112, pp. 35–47, Aug. 2019. doi: 10.1016/j.infsof.2019.04.005.
10. T. Menzies and T. Zimmermann, "Software analytics: What's next?" *IEEE Softw.*, vol. 35, no. 5, pp. 64–70, Sept. 2018. doi:10.1109/MS.2018.290111035.
11. G. Jeong, S. Kim, T. Zimmermann, and K. Yi, "Improving code review by predicting reviewers and acceptance of patches," Research on Software Analysis for Error-Free Computing Center, Seoul National Univ., Seoul, South Korea, Tech. Rep. ROSAEC MEMO 2009-006, Sept. 2009.
12. E. Shihab et al., "Studying re-opened bugs in open source software," *Empirical Softw. Eng.*, vol. 18, no. 5, pp. 1005–1042, 2013. doi: 10.1007/s10664-012-9228-6.
13. B. Lin, F. Zampetti, G. Bavota, M. D. Penta, M. Lanza, and R. Oliveto, "Sentiment analysis for software engineering: How far can we go?" in *Proc. 40th Int. Conf. Softw. Eng., (ICSE '18)*, New York: Association for Computing Machinery, 2018, pp. 94–104. doi: 10.1145/3180155.3180195.
14. B. Turhan, A. T. Mısırlı, and A. Bener, "Empirical evaluation of the effects of mixed project data on learning defect predictors," *Inf. Softw. Technol.*, vol. 55, no. 6, pp. 1101–1118, June 2013. doi: 10.1016/j.infsof.2012.10.003.
15. C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in *Proc. 2007 4th Int. Workshop on Mining Softw. Repositories (MSR'07:ICSE Workshops)*, p. 1. doi: 10.1109/MSR.2007.13.
16. H. Hu, H. Zhang, J. Xuan, and W. Sun, "Effective bug triage based on historical bug-fix information," in *Proc. 15th IEEE Int. Symp. Softw. Rel. Eng. (ISSRE)*, IEEE Computer Society, 2014, pp. 122–132. doi: 10.1109/ISSRE.2014.17.
17. M. Owhadi-Kareshk, S. Nadi, and J. Rubin, "Predicting merge conflicts in collaborative software development," in *Proc. ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas. (ESEM)*, Sept. 2019, pp. 1–11. doi: 10.1109/ESEM.2019.8870173.
18. F. Trautsch, S. Herbold, P. Makedonski, and J. Grabowski, "Addressing problems with replicability and validity of repository mining studies through a smart data platform," *Empirical Softw. Eng.*, vol. 23, no. 2, pp. 1036–1083, 2018. doi: 10.1007/s10664-017-9537-x.
19. J. Garcia, I. Krka, C. Mattmann, and N. Medvidovic, "Obtaining ground-truth software architectures," in *Proc. 2013 35th Int. Conf. Softw. Eng. (ICSE)*, pp. 901–910. doi: 10.1109/ICSE.2013.6606639.
20. K. A. Tecimer, E. Tüzün, H. Dibeklioglu, and H. Erdogmus, "Detection and elimination of systematic labeling bias in code reviewer recommendation systems," in *Proc. 2021 Eval. Assessment Softw. Eng., EASE*, New York: Association for Computing Machinery, pp. 181–190. doi: 10.1145/3463274.3463336.