Software Engineering for Data Analytics

Miryung Kim, University of California, Los Angeles

// We are at an inflection point where software engineering meets the data-centric world of big data, machine learning, and artificial intelligence. In this article, I summarize findings from studies of professional data scientists and discuss my perspectives on open research problems to improve data-centric software development. //

SOFTWARE ENGINEERING (SE) is currently meeting the data-centric discipline of artificial intelligence (AI), machine learning (ML), and big data. Almost on a daily basis, we hear about self-driving cars and drones enabled by AI, and companies hiring data scientists. Data analytics (DA) is in high demand, and the growth of DA-related hiring has more than doubled since 2014.[1]

Digital Object Identifier 10.1109/MS.2020.2985775
Date of current version: 18 June 2020

Similar to how bugs are problems in large software systems, defects could inevitably appear in data-centric software. In the case of Uber's self-driving vehicle, the consequence of inaccuracy was fatal. In March 2018, Elaine Herzberg was the first recorded case of a pedestrian fatality involving a self-driving autonomous car after a collision that occurred late in the evening.[2]

Although bugs in DA pose increasing risks, the SE research community somehow gravitated to applying data analytic techniques to SE problems, as opposed to enhancing SE techniques to improve data-centric development. In preparation for my keynote at the Automated Software Engineering (ASE) conference in 2019, I did a manual analysis of 285 papers numbering more than 10 pages from the last four years of ASE proceedings (2016–2019), categorizing each paper's problem and approach. I found that the percentage of papers that employ AI, ML, or big data has grown significantly from 2016 to 2019 (Figure 1). In fact, in 2019, there were more DA-related papers compared to the rest. However, most of these are about solving existing SE problems such as defect prediction, bug finding, document summarization, code recommendation, and testing using DA techniques such as deep learning, natural language processing, heuristic-based searches, multiobjective searches, classification, information retrieval, and so on, which I call data engineering for SE (DA4SE). Very few papers, only 13 out of 285 (4% of research papers at ASE 2016–2019) focused on improving SE for DA (Figure 1).

In this article, I make the case that we, the SE research community, should expand its research scope to extend and adapt existing SE to meet the new demands of data-centric software development and to improve the productivity of AI, ML, and big data engineers. I summarize findings from empirical studies of professional data scientists in collaboration with Microsoft Research.[3,4] In my opinion, key differences exist between traditional software development versus data-centric development, which makes it hard for software engineers to debug and test data-centric software or AI/ML-based software systems. I then share a few example research projects that

I have worked on with my students and collaborators that adapted existing software debugging and testing techniques to the domain of big data analytics.[4–10] I then sketch open research directions in SE for DA (SE4DA).

Data Scientists in Software Teams

We are at a tipping point where software companies are generating large-scale telemetry, machines, quality, and user data. Similar to how software developers and testers are established roles, data scientists are becoming a part of software teams. To understand what a data scientist is, what they do, and what challenges they face, we conducted the first in-depth interview study[3] as well as a large-scale survey.[4] We interviewed 16 data scientists and identified emerging themes from the transcripts, and clustered the themes. Then, to quantify and generalize their skills, working styles, tool usage, and challenges, we conducted a survey of nearly 800 data scientists. Figure 2 summarizes our two-phase study method and study participants.

The readers may ask, "What does a data scientist actually mean?" To deeply characterize this workforce, we clustered participants using a K-means algorithm based on their relative time spent on different activities. Nine categories emerged from the clustering analysis,[4] and the following three example categories are described:

- *Data shaper*: Data shapers spend a significant amount of time analyzing and preparing data. They have a higher representation of postgraduate degrees compared to the others. They are skilled in algorithms, ML, and numerical optimizations, but they are rather unfamiliar with front-end programming, which is required for the instrumentation of data collection. We named this category *data shapers*, because they extract and model relevant features from data.
- *Platform builder*: Platform builders spend 49% of their time developing platforms that instrument code to collect data. They have a strong background in big data distributed systems, back-end and front-end programming, and mainstream languages like C, C++, and C#. Platform builders identify as engineers who contribute to a data engineering platform and pipeline. They frequently mention the challenge of data cleaning.
- *Data analyzer*: Data analyzers often hold the job title of data scientist and are familiar with statistics, math, Bayesian statistics, and data manipulation. Many are R users and mention transforming data as a challenge.

Among all the categories of data scientists, when we asked, "How do you ensure correctness of your input and correctness of analytics?" many said that validation is a major challenge. Explainability is important: "To gain insights, you must go one level deeper." However, they expressed a general lack of confidence in analytics: "Honestly, we don't have a good method for this," and "just because the math is right, [it] doesn't mean that the answer is right."

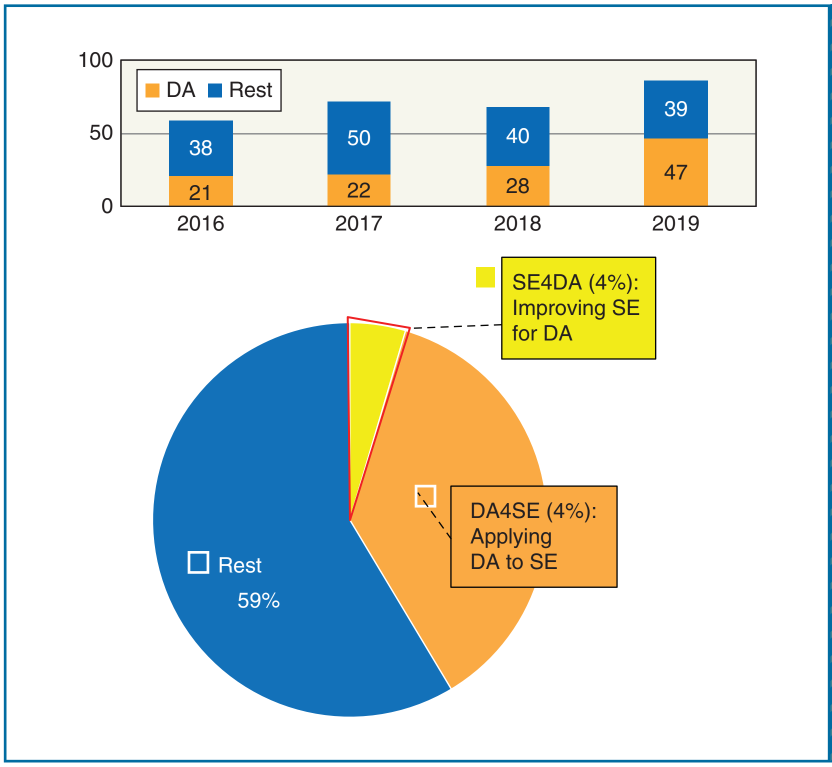
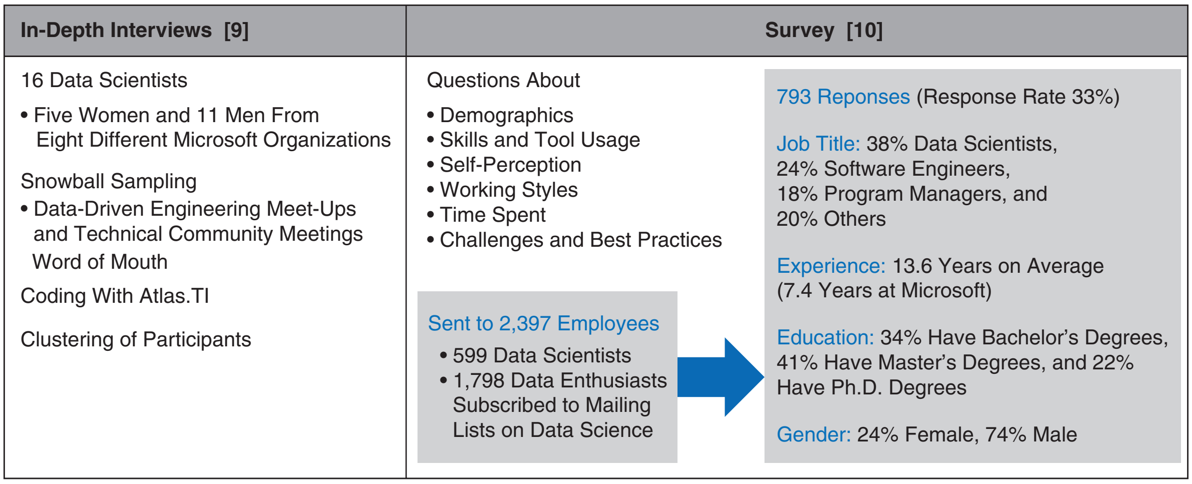


FIGURE 1. DA growth in SE. SE4DA is underinvestigated compared to data engineering for SE (DA4SE). (Source: ASE 2016–2019.)



In-Depth Interviews [9]	Survey [10]	
16 Data Scientists • Five Women and 11 Men From Eight Different Microsoft Organizations Snowball Sampling • Data-Driven Engineering Meet-Ups and Technical Community Meetings Word of Mouth Coding With Atlas.TI Clustering of Participants	Questions About • Demographics • Skills and Tool Usage • Self-Perception • Working Styles • Time Spent • Challenges and Best Practices Sent to 2,397 Employees • 599 Data Scientists • 1,798 Data Enthusiasts Subscribed to Mailing Lists on Data Science	793 Reponses (Response Rate 33%) Job Title: 38% Data Scientists, 24% Software Engineers, 18% Program Managers, and 20% Others Experience: 13.6 Years on Average (7.4 Years at Microsoft) Education: 34% Have Bachelor's Degrees, 41% Have Master's Degrees, and 22% Have Ph.D. Degrees Gender: 24% Female, 74% Male

FIGURE 2. The methodology used for studying professional data scientists' and participants' demographics.

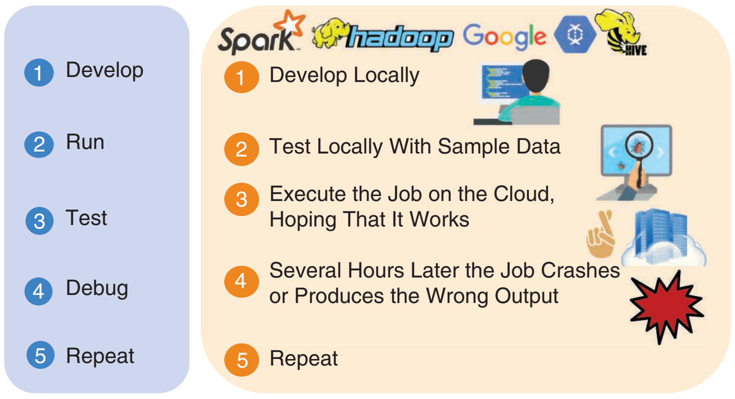

1 Develop
2 Run
3 Test
4 Debug
5 Repeat
Spark
hadoop
Google
HIVE
1 Develop Locally
2 Test Locally With Sample Data
3 Execute the Job on the Cloud, Hoping That It Works
4 Several Hours Later the Job Crashes or Produces the Wrong Output
5 Repeat

FIGURE 3. The traditional development versus big data analytics development.

How Is Traditional Development Different From Big Data Analytics Development?

In the previous section, I discussed how data scientists often have little confidence in their analytics software. By contrasting traditional development and data-centric development, I attempt to explain why data-centric software development is challenging (see Figure 3). This explanation is based on both our prior studies of data scientists[4] and other studies of ML development practices.[11,12] Data scientists develop an application and test it with samples using only a local machine. Then they execute this application on much larger data on a cluster. Several hours later, when the job crashes or produces a wrong or suspicious output, they repeat a trial-and-error debugging process. The following summarized differences contribute to the challenge of data-centric software development:

1. Data is huge, remote, and distributed.
2. Writing tests is hard. Developers often begin writing analytics without seeing the entire original input data, which are located in storage services such as Amazon S3. Because they write software based on a downloaded sample, which shows only an excerpt of the original data, it is difficult to write test oracles for the entire original input.
3. Failures are hard to define, in part due to a lack of tests and corresponding oracles.
4. System stacks are complex with little visibility because the underlying distributed systems and ML frameworks have complex scheduling, cluster management,

data partitioning, job execution, fault tolerance, and straggler management.

5. There is a gap between physical versus logical execution because analytics applications are highly optimized, lazily evaluated, and the user-defined application logic is interwoven with the execution of the framework code. For example, data-intensive scalable computing systems such as Spark provide execution logs of submitted jobs. However, these logs present only the physical view of big data processing, as they report the number of worker nodes, the job status at individual nodes, the overall job progress rate, the messages passed between nodes, and so on. These logs do not provide the logical view of program execution, for example, system logs do not convey which intermediate outputs are produced from which inputs, nor do they indicate what inputs are causing incorrect results or delays, and so forth.
6. Data tracing is hard. If there is a failure, it is hard to know which input contributed to which output because the current frameworks provide no traceability nor provenance support.

Debugging and Testing for Big Data Analytics

For the past five years, our team at the University of California, Los Angeles, has worked on extending and adapting software debugging and testing techniques to the domain of big data analytics written in Apache Spark.[4-10] From this experience, we have learned that designing interactive debugging primitives for a dataflow-based big data system requires a deep understanding of an internal execution model, job scheduling, and materialization; providing traceability requires reengineering a underlying data-parallel runtime framework; abstraction is a powerful force in simplifying code paths.

BigDebug: Interactive Debug Primitives for Big Data Analytics

We have had tools such as GDB (the GNU Project debugger) for a long time. So why is it hard to build an interactive debugger for Apache Spark? The naive implementation of breakpoints would not work because pausing the entire computation in the data-parallel pipeline reduces throughput, and it is clearly infeasible for a user to inspect billions of records through a regular watchpoint. BigDebug[6] does not pause program execution but instead simulates a breakpoint through on-demand state regeneration from the latest checkpoint and delivers program states in a guarded, stream-processing fashion. By effectively tapping into internal checkpointing and job-scheduling mechanisms, we were able to implement interactive debugging and repair capability in Apache Spark efficiently, while adding, at most, 34% overhead.[6]

Titian: Data Provenance for Apache Spark

Data provenance is a long-studied problem in databases. Given an output of query, data provenance identifies specific inputs contributing to the query results. The idea is similar to dynamic-taint propagation. For big data analytics with terabyte data, scalability poses a new challenge. To provide record-level data provenance, we reengineered Apache Spark's runtime by storing lineage tables (the input and output tag mappings) at a stage granularity in a distributed manner and by building a distributed optimized join for backward tracing, which is an order-of-magnitude faster than alternatives.[8]

BigSift: Automated Debugging of Big Data Analytics

BigSift takes a program and a test function as inputs and automatically finds a minimum subset of inputs producing test failures. BigSift combines two mature ideas, data provenance in database (DB) systems and delta debugging in SE, and implements several optimizations: 1) testing predicate pushdown, 2) prioritizing backward traces, and 3) bitmap-based memorization, which enabled us to build an automated debugging solution that is 66-times faster than delta debugging and takes 62% less time than the original job's run.[5]

BigTest: White-Box Testing of Big Data Analytics

Currently, developers sample data (for example, random sampling, top-n sampling, and top-k% sampling) to test DA, which leads to low code coverage. Another option is to use traditional test generation procedures such as symbolic execution, but such a technique would not scale for Apache Spark, which is roughly 700 KLOC.

To automatically generate tests for a Spark application, BigTest abstracts dataflow operators in terms of clean first-order logic.[7] For example, *join* could be defined as three equivalence classes where a key is only present in the left table, the right table, and neither. Then for a user-defined application code, BigTest performs symbolic execution and combines it together with dataflow logical specifications. These combined constraints are then solved using satisfiability modulo theories to create concrete inputs.

Only 30 or so records are required to achieve the same code coverage as the entire data, implying that testing on the entire data is not necessary. By automatically generating data with BigTest, we can reduce the required test data by 10^8, achieving nearly a 200-times speed-up.[7]

Open Research Directions in Data-Centric Development

This section discusses the open problems in SE4DA that have emerged from my observation of professional data scientists and my experience in researching debugging and testing techniques for big data analytics.[5–10]

Insight 1

We must expand the scope of debugging to include both code errors and data errors, and combine techniques in code and data repair. The SE community traditionally considers bugs as code defects, while the DB community considers bugs as data defects based on unexpected statistical distribution, functional dependencies, or schema mismatches. My perspective is that we need to combine insights from both communities to understand code errors and data errors in tandem. This is because data scientists write software systems based on an incomplete, partial understanding of input data, and thus, errors could exist in code that makes wrong assumptions about data, or new data could have drifted from the implicit assumptions made about the original input.

Consider the bug[7] that uses wrong delimiters such as splitting a string with "[]" instead of "\[\]," leading to a wrong output. A user may define this as a *data bug* or an *anomaly*, but it could be seen as a coding error based on the wrong assumptions made about the data. In fact, this error could be fixed by a code update, data cleaning, or both.

Performance debugging is as important as correctness debugging, and it requires enabling visibility into system stacks, code, and data.

Similar to how the SE community has worked on automated program repair and the DB community has worked on automated data cleaning and repair, now is the time to combine these insights to define what DA bugs mean and how to repair code errors and data errors together, as they are closely interrelated.

Insight 2

Performance debugging is as important as correctness debugging, and it requires enabling visibility into system stacks, code, and data. Based on our studies of data scientists, we found that the scope of debugging must go beyond functional correctness in the domain of big data analytics. Meeting performance requirements, which were often considered to be nonfunctional, secondary requirements, is as important as functional correctness.

Performance debugging, in particular, is often the biggest pain point for data analytics developers, as it depends on configuration, scaling, unbalanced tasks, IO, and memory-related issues in the cluster. A vertical stack is complex because it consists of a development environment, ML/AI libraries, runtimes, storage services, a Java virtual machine, containers, and virtual machines that also run heterogeneous hardware [for example, CPUs, GPUs, and FPGA (field programmable gate arrays)]. To diagnose and repair performance bottlenecks, we must consider the interaction between code, data, and system environments across a vertical stack. For example, debugging computational skews caused by interaction between code and a subset of data requires tracking latency information for individual inputs throughout various computational stages.[10]

Insight 3

We must design easy-to-use, easy-to-extend oracle-specification techniques for debugging and testing heuristics-based, probabilistic, and predictive analytics. Creating oracles for heuristics-based, probabilistic, and predictive DA is different from how we define oracles in traditional unit testing. Metamorphic testing relates changes between two inputs to changes between two corresponding outputs.[13] Existing techniques for testing neural networks use metamorphic testing, but they are limited to checking whether input perturbations still produce the same classification results and test only an equivalence-based metamorphic relation.

Insight 4

We must design new debugging techniques that quantify the degree of influence and importance between input distributions and unexpected behavior. Traditionally, debugging techniques such as delta debugging attribute the cause of test failures to individual failure-inducing inputs equally. We must quantify the notion of importance when debugging faulty inputs, as a bug is often caused by a subset of input data near decision boundaries, a particular data partition, or a particular input distribution drifted from the original data assumption, as opposed to a single input. For example, training-set debugging in ML identifies a subset of inputs, leading to mis-classifications using the mathematical notion of influence functions[14] by isolating input data near decision boundaries. We must leverage such ideas to extend and adapt existing software debugging to data-centric software.

By studying professional data scientists, and based on the experience of adapting SE techniques to debug and test big data applications, I have found that data-centric software development is different from traditional software development in several ways. To support data-centric software development, we must investigate how code errors and data errors interact, and we should not limit the scope of debugging to correctness debugging because performance debugging is as important as correctness debugging to many data scientists. Inherently, it is challenging to define what should be a correct behavior for heuristics-based, probabilistic, and predictive analytics. Therefore, we must design easy-to-extend, easy-to-use specification techniques to facilitate debugging and testing. Solving these open problems requires the SE community to work together with the AI, ML, systems, and DB communities.

ABOUT THE AUTHOR

MIRYUNG KIM is a full professor in the Department of Computer Science at the University of California, Los Angeles. Her research interests include code clones and code duplication detection, management, and removal solutions. She has taken a leadership role in defining the emerging area of software engineering for data science. Kim received a Ph.D. in computer science and engineering from the University of Washington, Seattle. She received various awards including an NSF CAREER Award, a Microsoft Software Engineering Innovations Foundation Award, a Google Faculty Research Award, and an Okawa Foundation Research Award. She is the program cochair of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering 2022 and the International Conference on Software Maintenance and Evolution 2019 and is an associate editor of *IEEE Transactions on Software Engineering*. Contact her at miryung@cs.ucla.edu.

Acknowledgments

I thank my collaborators Thomas Zimmermann, Rob Decline, and Andrew Begel for their joint work on the study of data scientists. I also thank University of California, Los Angeles students and collaborators Tyson Condie, Aria Emoji, Muhammad Ali Gulzar, Matteo Interlandi, Shaghayegh Mardani, Todd Millstein, Madanlal Musuvathi, Kshitij Shah, Sai Deep Tetali, Jason Jia Teoh, Seunghyun Yoo, and Harry Xu for their automated debugging and testing of Apache Spark. I thank my Ph.D. student Gulzar for being a sounding board for this software engineering for data analytics journey.

This work is in part supported by National Science Foundation award 1764077.

References

1. D. Culbertson "High demand for data science jobs," Indeed Hiring Lab, Mar. 15, 2018. [Online]. Available: https://www.hiringlab.org/2018/03/15/data-science-job-postings-growing-quickly
2. Wikipedia, "Death of Elaine Herzberg," Apr. 4, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Death_of_Elaine_Herzberg
3. S. Amershi et al, "Software engineering for machine learning: A case study," in *Proc. 2019 IEEE/ACM 41st Int. Conf. Software Engineering: Software Engineering Practice (ICSE-SEIP)*, May 2019, pp. 291–300. doi: 10.1109/ICSE-SEIP.2019.00042.
4. M. A. Gulzar, M. Interlandi, X. Han, M. Li, T. Condie, and M. Kim, "Automated debugging in data-intensive scalable computing," in *Proc. 2017 Symp. Cloud Computing (SoCC '17)*. New York: ACM, 2017, pp. 520–534. doi: 10.1145/3127479.3131624.
5. M. A. Gulzar et al., "Bigdebug: Debugging primitives for interactive big data processing in spark," in *Proc. 38th Int. Conf. Software Engineering* (ICSE '16). New York: ACM, 2016, pp. 784–795. doi: 10.1145/2884781.2884813.
6. M. A. Gulzar, S. Mardani, M. Musuvathi, and M. Kim, "White-box testing of big data analytics

with complex user-defined function," in *Proc. 2019 27th ACM Joint Meeting European Software Engineering Conf. and Symp. the Foundations of Software Engineering (ESEC/FSE 2019)*. New York: ACM, 2019, pages 290–301. doi: 10.1145/3338906.3338953.

7. M. Interlandi et al., "Adding data provenance support to apache spark," *VLDB J.*, vol. 27, no. 5, pp. 595–615, Aug. 2017. doi: 10.1007/s00778-017-0474-5.
8. M. Interlandi et al., "Optimizing interactive development of data-intensive application," in *Proc. Seventh ACM Symp. Cloud Computing*, Santa Clara, CA, Oct. 5–7, 2016, pp. 510–522. 2016. doi: 10.1145/2987550.2987565.
9. M. Kim, T. Zimmermann, R. DeLine, and A. Begel, "The emerging role of data scientists on software development teams," in *Proc. 38th Int. Conf. Software Engineerin (ICSE '16)*. New York: ACM, 2016, pages 96–107. doi: 10.1145/2884781.2884783.
10. M. Kim, T. Zimmermann, R. DeLine, and A. Begel, "Data scientists in software teams: State of the art and challenges," *IEEE Trans. Softw. Eng.*, vol. 44, no. 11, pp. 1024–1038, Nov. 1, 2018. doi: 10.1109/TSE.2017.2754374.
11. P. W. Koh and P. Liang, "Understanding black-box predictions via influence functions," in *Proc. 34th Int. Conf. Machine Learning (ICML-17)*, vol. 70. New York: ACM, 2017, pp. 1885–1894.
12. D. Sculley et al., "Hidden technical debt in machine learning systems," in *Advances in Neural Information Processing Systems*, vol. 28, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Red Hook, NY: Curran Associates, Inc., 2015, pp. 2503–2511.
13. S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortes, "A survey on metamorphic testing," *IEEE Trans. Softw. Eng.*, vol. 42, no. 9, pp. 805–824, Sept. 2016. doi: 10.1109/TSE.2016.2532875.
14. J. Teoh, M. A. Gulzar, G. H. Xu, and M. Kim, "Perfdebug: Performance debugging of computation skew in dataflow system," in *Proc. ACM Symp. Cloud Computing (SoCC '19)*, pp. 465–476. New York: ACM, 2019. ACM. doi: 10.1145/3357223.3362727.

Digital Object Identifier 10.1109/MS.2020.2996915