The AI Effect: Working at the Intersection of AI and Software Engineering
Editor: sw4-2020@computer.org

SE4DA—Software Engineering for Data Analytics

Miryung Kim
University of California, Los Angeles

***Abstract*—We are at an inflection point where software engineering meets the data-centric world of big data, machine learning, and artificial intelligence. As software development gradually shifts to data analytics development with AI and ML technologies, existing software engineering techniques must be re-imagined to provide the productivity gains that developers desire. In this article, I summarize findings from our studies of professional data scientists—what a data scientist is, what data scientists do, and what challenges they face. I discuss a few examples of my group's research projects that adapted existing debugging and testing techniques to the domain of big data analytics. Based on these experiences, I discuss my perspectives on open research problems to improve the productivity of data-centric software development.**

Software engineering (SE) is currently meeting the data-centric discipline of AI, 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 are in high demands and the growth of data analytics related hiring has more than doubled since 2014.[1]

Similar to how bugs are problems in large software systems, defects could inevitably appear in data-centric software. In 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 that evening.[2]

While bugs in data analytics pose increasing risks, the SE research community somehow gravitated to applying data analytic techniques to software engineering problems, as opposed to enhancing software engineering techniques to improve data-centric development. In preparation for my keynote at Automated Software Engineering in 2019, I did a manual analysis of 285 papers over 10 pages from the last 4 years of ASE proceedings (2016 to 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 are more data analytics related papers, compared to the rest. However, most of these are about solving

[1] https://www.hiringlab.org/2018/03/15/data-science-job-postings-growing-quickly

[2] https://en.wikipedia.org/wiki/Death_of_Elaine_Herzberg

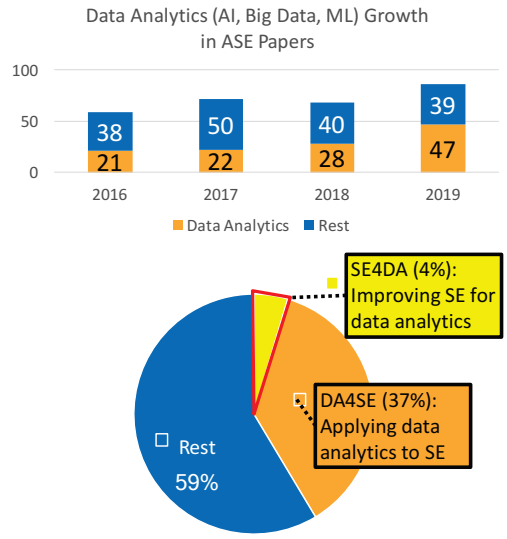

Figure 1. Data Analytics Growth in SE (ASE 2016 to 2019). SE4DA is under-investigated, compared to DA4SE.

existing SE problems such as defect prediction, bug finding, document summarization, code recommendation, and testing, using data analytics techniques such as deep learning, NLP, heuristic-based search, multi-objective search, classification, information retrieval, etc. Very few papers, only 13 out of 285 papers (4% of research papers in ASE 2016-2019) focused on improving SE for DA (Figure 1).

In this article, I hope to make a case that we, the software engineering research community, should expand its research scope to extend and adapt existing software engineering to meet the new demands of data-centric software development and to improve the productivity of AI, ML, and Big Data engineers. I will summarize findings from empirical studies of professional data scientists in collaboration with Microsoft Research [7, 8]. In my opinion, key differences exist between traditional software development vs. data centric development, which makes it hard for software engineers to debug and test data-centric software or AI/ML-based software systems. I will 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 [3, 2, 4, 12, 5, 6]. I will then sketch open research directions in SE4DA that are based on my personal experience and observations of professional data scientists.

Data scientists in software teams

We are at a tipping point where software companies are generating large-scale telemetry, machine, 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 [7] and a large scale survey [8]. We interviewed 16 data scientists and identified emerging themes from the transcripts, clustered the themes. Then to quantify and generalize their skills, working styles, tool usage, and challenges, we conducted a survey with almost 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 relative time spent on different activities. Nine categories emerged from the clustering analysis [8] and below describes three example categories.

- **Data Sharper**: Data shapers spend a significant amount of time in analyzing and preparing data. They have a higher representation of post-graduate degrees compared to the rest. They are skilled in algorithms, machine learning, and numerical optimizations, but they are rather unfamiliar with front end programming, required for instrumentation for data collection. We named this category as data shapers, because they extract and model relevant features from data.
- **Platform Builder**: Platform builders develop platforms to instrument code to collect data, spending 49% of time. They have a strong background in big data distributed systems, back-end and front-end programming, and main stream 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 are often hold a job title as a data scientist and are familiar

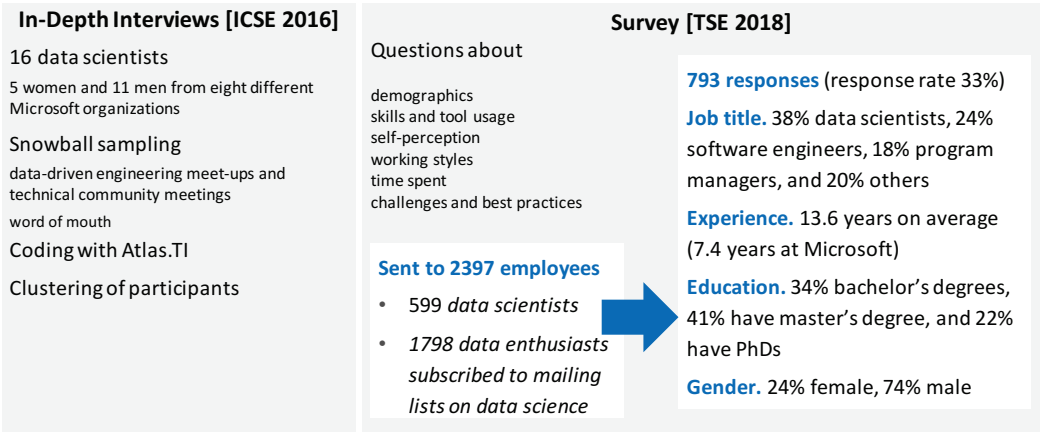

Figure 2. Methodology for Studying Professional Data Scientists and Participants Demographic

with statistics, math, Bayesian statistics, and data manipulation. Many are R users and mention transforming data as a challenge.

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

How is traditional development different from big data analytics development?

In the previous section, I discussed how data scientists often have little confidence about their analytics software. By contrasting traditional development and data-centric development, I will try to explain why data-centric software development is challenging. This explanation is based on both our prior studies of data scientists [8] and other studies on ML development practices [10, 1]. Data scientists develop an application and test it with only samples using 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. Differences summarized below contribute to the challenge of data-centric software development.

1) **Data is huge, remote, and distributed**.
2) **Writing tests is hard**. Developers often start writing analytics without seeing the entire, original input data, located in storage services such as Amazon S3. Because they write software based on a downloaded sample, which shows only the 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 lack of tests and corresponding oracles.
4) **System stacks are complex** with little visibility, because 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 the physical vs. 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, DISC 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, etc. These logs do not provide the logical view of program execution e.g., system logs do not convey which intermediate outputs are produced from which inputs, nor do they indicate what inputs are causing incorrect

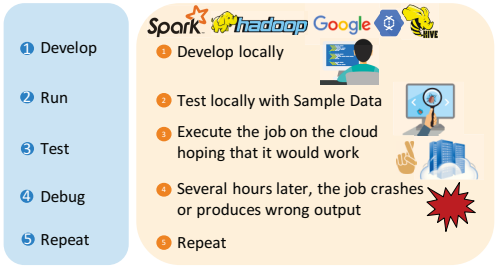

Figure 3. Traditional development vs. big data analytics development

results or delays, etc.

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 UCLA have worked on extending and adapting software debugging and testing techniques to the domain of big data analytics written in Apache Spark [3, 2, 4, 12, 5, 6]. From this experience, we have learned that designing interactive debug primitives for a dataflow based big data system requires deep understanding of an internal execution model, job scheduling, and materialization; providing traceability requires re-engineering a underlying data-parallel runtime framework; and abstraction is a powerful force in simplifying code paths.

BigDebug: Interactive Debug Primitives for Big Data Analytics

We have had tools such as GDB for a long time. So why is hard to build an interactive debugger for Apache Spark? Naïve 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 billion of records through a regular watchpoint. BigDebug [3] 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 [3].

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 re-engineered Apache Spark's runtime by storing lineage tables (the input and output tag mappings) at a stage granularity in a distributed manner and building a distributed optimized join for backward tracing, which is order of magnitude faster than alternatives [5].

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 DB and delta debugging in SE—and implements several optimizations: (1) test predicate push-down, (2) prioritizing backward traces, and (3) bitmap based memorization, which enabled us to build an automated debugging solution that is 66X faster than delta debugging and takes 62% less time than the original jobs run [2].

BigTest: White-Box Testing of Big Data Analytics

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

To automatically generate tests for a Spark application, BigTest abstracts dataflow operators, in terms of clean first order logic [4]. 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. Then these combined constraints are solved using SMT 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 almost 200X speed up [4].

Open research directions in data-centric development

This section discusses open problems in SE4DA. In this section, I describe several open research directions in SE4DA, which emerged from my observation of professional data scientists [7, 8] and my research experience in automated debugging and test input generation for big data analytics [3, 2, 4, 12, 5, 6].

> 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 reason about code errors and data errors in tandem. This is because data scientists write software systems based on 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 reported in [4] that uses wrong delimiters such as splitting a string with "[ ]" instead of "\[\]" leading to a wrong output (https://stackoverflow.com/questions/52083828). A user may define this as a data bug or 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 code update, data cleaning, or both.

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 data analytics bugs mean and how to repair code errors and data errors together, as they are closely inter-related.

> Insight 2: Performance debugging is as important as correctness debugging and performance debugging 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 as non-functional, secondary requirements—is as important as functional correctness.

Performance debugging, in particular, is often the biggest painpoint 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, as it consists of a development environment, ML/AI libraries, runtimes, storage services and JVM, containers and VMs that also run heterogeneous hardware (e.g., CPU, GPU and FPGA). To diagnose and repair performance bottlenecks, we must consider 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 [12].

> 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 data analytics is different from how we define oracles in traditional unit testing. Metamorphic testing relates changes between two inputs to changes between two corresponding outputs [11]. Existing techniques for testing neural networks use metamoprhic testing, but they are limited to checking whether input perturbations still produce the same classification results, testing an equivalence-based metamorphic

relation only.

> 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 mathematical notion of *influence functions* [9] by isolating input data near decision boundaries. We must leverage such idea to extend and adapt existing software debugging to data-centric software.

Conclusion

By studying professional data scientists and based on the experience of adapting SE techniques to debug and test big data applications, I 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, as 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.

Acknowledgment

I thank my collaborators—Thomas Zimmermann, Rob Decline, and Andrew Begel—for the joint work on the study of data scientists. I thank UCLA students and collaborators—Tyson Condie, Aria Emoji, Muhammad Ali Gulzar, Matteo Interlandi, Shaghayegh Mardani, Todd Millstein, Madanlal Musuvathi, Kshitij Shah, Sai Deep Tetali, and Seunghyun Yoo for automated debugging and testing for Apache Spark. I thank my student Gulzar for being a sounding board for this SE4DA journey.

REFERENCES

1. S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 291–300, May 2019.
2. M. A. Gulzar, M. Interlandi, X. Han, M. Li, T. Condie, and M. Kim. Automated debugging in data-intensive scalable computing. In *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, pages 520–534, New York, NY, USA, 2017. ACM.
3. M. A. Gulzar, M. Interlandi, S. Yoo, S. D. Tetali, T. Condie, T. Millstein, and M. Kim. Bigdebug: Debugging primitives for interactive big data processing in spark. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 784–795, New York, NY, USA, 2016. ACM.
4. M. A. Gulzar, S. Mardani, M. Musuvathi, and M. Kim. White-box testing of big data analytics with complex user-defined functions. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, pages 290–301, New York, NY, USA, 2019. ACM.
5. M. Interlandi, A. Ekmekji, K. Shah, M. A. Gulzar, S. D. Tetali, M. Kim, T. Millstein, and T. Condie. Adding data provenance support to apache spark. *The VLDB Journal*, Aug 2017.
6. M. Interlandi, S. D. Tetali, M. A. Gulzar, J. Noor, T. Condie, M. Kim, and T. D. Millstein. Optimizing interactive development of data-intensive applications. In *Proceedings of the Seventh ACM Symposium on Cloud Computing, Santa Clara, CA, USA, October 5-7, 2016*, pages 510–522, 2016.

7. M. Kim, T. Zimmermann, R. DeLine, and A. Begel. The emerging role of data scientists on software development teams. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 96–107, New York, NY, USA, 2016. ACM.
8. M. Kim, T. Zimmermann, R. DeLine, and A. Begel. Data scientists in software teams: State of the art and challenges. *IEEE Transactions on Software Engineering*, pages 1–17, 2017.
9. P. W. Koh and P. Liang. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML17, page 18851894. JMLR.org, 2017.
10. D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison. Hidden technical debt in machine learning systems. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2503–2511. Curran Associates, Inc., 2015.
11. S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortes. A survey on metamorphic testing. *IEEE Transactions on Software Engineering*, 42(09):805–824, sep 2016.
12. J. Teoh, M. A. Gulzar, G. H. Xu, and M. Kim. Perfdebug: Performance debugging of computation skew in dataflow systems. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '19, pages 465–476, New York, NY, USA, 2019. ACM.