Research Opportunities for the Big Data Era of Software Engineering

Robert DeLine
Microsoft Research
Redmond, WA, USA
rdeline@microsoft.com

***Abstract*—Big Data Analysis is becoming a widespread practice on many software development projects, and statisticians and data analysts are working alongside developers, testers and program managers. Because data science is still an emerging discipline in software projects, there are many opportunities where software engineering researchers can help improve practice. In terms of productivity, data scientists need support for exploratory analysis of large datasets, relief from clerical tasks like data cleaning, and easier paths for live deployment of new analyses. In terms of correctness, data scientists need help in preserving data meaning and provenance, and non-experts need help avoiding analysis errors. In terms of communication and coordination, teams need more approachable ways to discuss uncertainty and risk, and support for data-driven decision making needs to become available to all roles. This position paper describes these open problems and points to ongoing research beginning to tackle them.**

I. Big Data in Software Engineering

The collection and analysis of large-scale data is becoming a widespread practice among software development teams. The big data era of software engineering arguably began two decades ago. Early search engines, like Google's, based their services on the analysis of millions of web documents. These engines, as well as early online retailers, like Amazon, pioneered the collection of usage data to mine patterns of customer behavior. Other companies used big data behind the scenes to improve product quality, like Microsoft's Windows Error Reporting, which collects crashes to allow automatic bucketing and triage [1].

Today, the adoption of cloud computing is turning big data analysis into a core competency in software engineering. Many teams are welcoming statisticians and data analysts (collectively called *data scientists*) alongside existing roles like developers, testers, and program managers. For software engineering researchers, this transition creates the opportunity to study, understand and improve these practices. This position paper describes some of these research opportunities, based both on recent studies of data scientists [2] [3] and discussions at Microsoft about careers in data science.

To discuss these opportunities, it helps to establish the landscape of how development teams use big data. We can characterize current practice along several dimensions: the data domain; the analysis goal; and the supported stakeholders and activities. Development teams currently collect data in several domains, including:

- data about customers (e.g. usage logs, product purchases, game play);
- data about execution behavior (e.g. crash data, performance metrics, server loads); and
- data about team work practices (e.g. the bug database, the source repository).

The purpose of analyzing data is to surface insights to improve some aspect of the software, including:

- the software's user experience (e.g. enhancing data-driven features like search results, experimentally releasing new features through A/B testing);
- the software's execution (e.g. reducing crashes, or deciding when to scale out to more servers); and
- the team's development process (e.g. closing bugs faster, assigning testing resources)

Analyses also vary in the stakeholders and activities that they support:

- ad hoc information seeking (e.g. a program manager testing a hypothesis about feature usage, or a developer analyzing crash data to find a bug's root cause);
- decision making (e.g. a team presentation about the results of A/B testing a potential feature, or a "war room" tracking open bugs to decide when to ship);
- online monitoring (e.g. a dashboard of signals about software usage, or a display of status lights about server availability); and
- product features (e.g. a player-matching algorithm in a game, or advertisements served on web pages).

Some parts of this space are already well supported. For example, analyzing customer data to support decision making, called *business intelligence*, has been the subject of database research and product development for many years. Other parts of the space, for example, those involving online monitoring, are newer and enjoy less tool support. This paper describes some of the areas where researchers are currently working and where further research would help.

II. Research Opportunities in Productivity

Back in the era of mainframe computing, software developers had separate tools for each stage of development (compilation, linking, loading, performance analysis, debugging), which coordinated by passing files from one program to another. Many of these tools operated in a batch mode, which

978-1-4673-7025-7/15 $31.00 © 2015 IEEE
DOI 10.1109/BIGDSE.2015.13

meant long waits for any output. Sadly, today's tools for data scientists are reminiscent of this bygone era [2] [3]. To turn a huge dataset into an actionable result, a data scientist typically cobbles together a workflow from a suite of tools, including map-reduce frameworks like Hadoop, scripting software like R or Python, visualization tools like Tableau, and spreadsheets like Excel. They pass data from one tool to another through intermediate files, and many of these tools operate in a batch fashion.

A. Supporting Exploratory Behavior

No one likes waiting, but the batch-style operation in today's analysis tools is especially harmful for data scientists. Data scientists often operate in an exploratory mode, sometimes because they are pursuing speculative goals (e.g. evaluating opportunities for new features), sometimes because they are reacting to the contents of the data (e.g. exploratory data analysis [4]). Needless to say, being exploratory is next to impossible when each step takes hours to complete. Data scientists currently work around the delays by exploring small data samples before repeating the work on the whole dataset. The easiest samples to acquire are typically biased, for example, the first 10,000 lines of the file or today's batch of telemetry data. Unfortunately, results from biased samples can be misleading. So, an exploration that seems promising on the sample may be unprofitable on the whole dataset, or vice versa.

Because of the obvious downside of long waits, there is active research in interactive query engines. Tools like Dremel [5] and PowerDrill [6] provide interactivity through fast processing speed, using techniques like columnar representations and storing data in memory. Other tools like DBO [7] and Stat [8] instead provide progressive computation, where the tool provides partial results that are updated as more data are processed. In the case where the data are progressively processed in random order, the data constitute a steadily increasing population sample, and the tool can present steadily decreasing error bounds [9].

B. Reducing Clerical Work

So far, data scientists do not enjoy the convenience of the equivalent of an integrated development environment. Some environments, like IPython, cover several steps, but many workflows require the data scientist to coordinate several tools, often using intermediate files. As a result, a data analysis project is often a messy collection of files with implicit relationships. Data scientists currently use vigilance to keep their analysis materials organized. Without careful attention to detail, it can be impossible to distribute an analysis to colleagues or to resurrect it after months aways from it; bits and pieces will be missing. Researchers have explored tools for automatically bundling analysis materials [10]. Nonetheless, the root cause, namely coordinating multiple tools with their own notations and user interfaces, is an ongoing problem.

Some steps in a data analysis are themselves tedious and clerical, particularly "data cleaning," that is, the initial steps of transforming data from its format at collection to a format suitable for algorithmic analysis. The "dirtiness" of raw data reflects the complexities of data collection, including faulty collectors that lose data, human errors in data entry, and inconsistencies in the data domain (e.g. most people have surnames, but some do not). Because these issues are erratic, human intervention is needed. Researchers are working on mixed-initiative tools for these clean-up tasks, in which software amplifies the human effort [11] [12]. Of course, the ideal would be data cleaning with no human intervention.

C. Smoothing the Path from Exploration to Deployment

Increasingly, teams collect data on a continuous basis, often called *telemetry*. They use telemetry data to compute metrics, called *key performance indicators*, which are monitored through periodic reports or live dashboards. Of course, over time, a team's software changes, customer behavior changes, and the business environment changes. In response, the team will want to change what it monitors. This suggests an iterative development cycle, which Chandramouli *et al.* call monitor-mine-manage (M3) [13]. First, a data scientist does ad hoc explorations of historical telemetry data to invent new metrics. To implement these new metrics, the team updates the deployed software's data collection and analysis logic. Finally, the team monitors the new metrics, creating a new status quo that starts the cycle again.

To carry out this cycle, a series of ad hoc queries eventually becomes an operational part of the system. We can schematically represent the analysis logic's migration path with the following quadrants:

	speculative analysis	deployed analysis
historical data	A	B
live data	C	D

The M3 path takes an analysis from quadrant A to quadrant D. Preferably, this migration path would go through quadrant C, to allow the analysis to be tested first in a live setting before deployment.

Ideally, the M3 cycle would be as routine and painless as the edit-compile-debug cycle, but this is far from reality yet. In a typical setup today, the live source of telemetry data and the system that archives the data are often entirely separate, with different schemas, access policies, and query languages. So, any analysis logic written for the archival data would have to be rewritten to be deployed in the live system. Indeed, dividing the migration into two steps (A to C, C to D) may require two rewritings.

III. Research Opportunities in Correctness

The results of analyzing big data often have serious consequences. For instance, a team might analyze its bug database to decide if their product is ready to ship, or it might analyze usage data to choose which features to implement in the next release. If such an analysis were wrong, the team could make poor choices, causing the project to fail. Similarly, when

analysis results are exposed as software features, poor analyses can lead to poor customer experiences, again threatening the success of the project. Hence, researchers have the opportunity to help software teams be confident about their data analysis results.

A. Data Meaning and Provenance

Developers complain about the difficulty of understanding code that others have written, which prompts the academic study of program comprehension. Similarly, data scientists complain about the difficulty of understanding data that others have collected, manipulated and organized. (Is there an opportunity for a field of data comprehension?)

We can divide issues about understanding data into two distinct phases: data *meaning*, about how data are connected to phenomena in the world that they record; and data *provenance*, about how data are computed from other data. Misunderstandings about meaning and provenance can lead to incorrect analysis results. To prevent errors, the standard today is for each person and tool touching the data to provide meticulous records. Researchers from the life sciences are creating standards for this record keeping, like the Open Provenance Model [14].

Understanding when data are "correct" is a different problem than program correctness. (Indeed, many verification regimens abstract away the details of data, for example, the predicates in Hoare logic.) In the absence of a correctness criterion for a data source, one approach is to treat the data's typical distribution as an implicit contract and to look for anomalies [15]. Another approach to assuring the data's meaning would be turn domain knowledge into checks. In the same way 'assert' statements are a popular lightweight technique for capturing and checking expectations about execution behavior, perhaps there are similar lightweight ways for data scientists to capture expectations about their data.

In terms of provenance, we need techniques for reasoning about computations that transform data. There is emerging research on checking probabilistic assertions in programs that compute over uncertain data [16]. This involves propagating statistical information through a program's data flow, which would be useful in analysis scripts. Such approaches are an important step toward an overall goal of reasoning in a statistically rigorous way through an entire workflow, from the point at which data are recorded, through all transformations in the data analysis, to an actionable conclusion.

Software systems evolve; therefore, the data about those systems also evolve. These changes over time make reasoning about the data even more difficult. For instance, a software project's code is often interlaced with logging statements that produce data about user and execution behavior. As developers change the code, they also change the logging statements, which in turn changes the data available for analysis. Unless these changes to the log format are explicitly tracked, it can be easy to draw wrong conclusions. For example, when an action is missing in the logs, did the action not occur or did it occur before logging statements captured that type of action? In short, tools to analyze telemetry data must carefully handle telemetry evolution.

B. Provide "Guard Rails" for Non-experts

Ideally, every development team would have an expert statistician to ensure that the team's decisions are based on high-confidence analyses. Realistically, there will not be enough statisticians to satisfy the demand, so non-experts will carry out at least some analyses. This creates the opportunity for end-user data science, analogous to end-user programming. The standard today is self-service business intelligence, which relies on a division of labor between experts and non-experts. An expert data analyst schematizes the data and creates indices and aggregations (data cubes), to provide a space of possible queries for business users to run. While this approach is useful, it limits exploration. A program manager, for example, might like to understand user preferences by supplementing her team's schematized data with unstructured data, like Tweets mentioning the product.

The early stages of a data science workflow, like cleaning, searching, and aggregating, tend to be more approachable, because they deal directly with the data and leverage skills from spreadsheets. Later stages, like engineering features for training a classifier or testing a correlation, require expertise. Nonetheless, non-experts likely understand the goals of these steps, e.g. separating classes of data or determining whether a relationship exists. This suggests that it may be possible to provide high-level commands for these steps, so long as there are sufficient checks to ensure that the steps are sensible. When fully integrated environments for data analysis emerge, the integration will allow analyses to be tracked end-to-end, from raw data to final results. This provides further opportunity for error checking.

Looking for problems in a data analysis is a less black-and-white problem than finding bugs in software. The safety properties checked by verification and bug-finding tools are indisputable: no one wants a null dereference or memory corruption. The "rules" of data analysis are more heuristic. For example, many statistical tests have preconditions about the distribution of the data, e.g. normality. However, tests for normality have thresholds, which require human judgment. Hence, an attempt to "automate away" statistical knowledge, for example, by fitting empirical data to probability distributions and choosing compatible tests, would be overly preemptive. Instead, tools should allow non-experts to express high-level data modeling choices based on their own insights, while informing the user about the repercussions of those choices.

IV. Research Opportunities in Communication and Collaboration

A. Conveying Risk and Uncertainty

The purpose of many data analyses is to help teams make critical decisions. Part of the decision-making process is weighing the credibility of the insights from data analysis. For analyses whose inputs are uncertain, decision makers also

need to understand this uncertainty and its risks. Probability and statistics provide precise ways to convey these concepts, but are difficult even for experts to interpret. Recently, even classic validity measures, like p-values, have become open to debate [17].

Of course, this communication problem exists for anyone using big data to make decisions, but the context of software engineering may allow opportunities for approaches that would not generally apply. For example, software teams are skillful at computational thinking, which means that simulation-based techniques, like Monte Carlo methods, may provide a basis for clearer communication.

B. Supporting Broad Participation

For most of the history of software engineering, developers and testers were seen as different roles, with different knowledge, skills, and career paths. However, with the rise of unit testing and test-driven development, the boundary between the roles has blurred. Today, developers themselves often write tests, both to ensure continuous software quality and to drive the design of programming interfaces. Testing has become less of a distinct role and more of a skill set that multiple roles possess.

Similarly, data science today is seen as a distinct role, with its own knowledge, skills and career path. Indeed, data mining, machine learning, and statistics are highly prized in the current job market. Will data science remain a specialty, or like testing, will some of its skills and knowledge spread to other roles? The latter seems likely, since activities like ad hoc information seeking are universal. Program managers, for instance, have questions about customer behavior. Testers have questions about crashes and the frequency of execution paths. Developers are curious about how their features are used. Since data collection is not free, the team will need to coordinate the information needs of different roles.

V. Conclusion

This paper presents a handful of difficult problems that software teams face as they exploit big data analysis, with references to some ongoing research to address these problem. Other existing big data problems are important, but omitted because they are universal to anyone working with big data, not just software teams. Examples include data security, privacy, and ethical uses of customer data. As software engineering researchers tackle their community's problems, they will hopefully participate in the larger conversation about these other issues.

Tackling these open problems could fundamentally alter how software development teams make decisions and create new ideas. As Microsoft CEO Satya Nadella wrote in his blog, "We believe that with the right tools, insights can come from anyone, anywhere, at any time. When that happens, organizations develop what we describe as a 'data culture.'" [18] Such a data culture would allow each team member to make evidence-based choices in his or her individual role, allow the team to make informed tradeoffs and decisions, and allow plans to be made against predictions with measurable uncertainty.

References

[1] K. Glerum, K. Kinshumann, S. Greenberg, G. Aul, V. Orgovan, G. Nichols, D. Grant, G. Loihle, and G. Hunt, "Debugging in the (very) large: ten years of implementation and experience," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. ACM, 2009, pp. 103–116.

[2] S. Kandel, A. Paepcke, J. M. Hellerstein, and J. Heer, "Enterprise data analysis and visualization: An interview study," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 12, pp. 2917–2926, 2012.

[3] D. Fisher, R. DeLine, M. Czerwinski, and S. Drucker, "Interactions with big data analytics," *ACM interactions*, vol. 19, no. 3, pp. 50–59, 2012.

[4] J. W. Tukey, *Exploratory data analysis*. Reading, Mass., 1977.

[5] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, "Dremel: Interactive analysis of web-scale datasets," in *Proc. of the 36th Int'l Conf on Very Large Data Bases*, 2010, pp. 330–339. [Online]. Available: http://www.vldb2010.org/accept.htm

[6] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "Mapreduce online." in *NSDI*, vol. 10, no. 4, 2010, p. 20.

[7] C. Jermaine, S. Arumugam, A. Pol, and A. Dobra, "Scalable approximate query processing with the dbo engine," *ACM Transactions on Database Systems (TODS)*, vol. 33, no. 4, p. 23, 2008.

[8] M. Barnett, B. Chandramouli, R. DeLine, S. Drucker, D. Fisher, J. Goldstein, P. Morrison, and J. Platt, "Stat!: An interactive analytics environment for big data," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 2013, pp. 1013–1016.

[9] D. Fisher, I. Popov, S. Drucker *et al.*, "Trust me, i'm partially right: incremental visualization lets analysts explore large datasets faster," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2012, pp. 1673–1682.

[10] P. J. Guo and M. Seltzer, "Burrito: Wrapping your lab notebook in computational infrastructure," in *Proceedings of the 4th USENIX Workshop on the Theory and Practice of Provenance*, ser. TaPP'12. Berkeley, CA, USA: USENIX Association, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2342875.2342882

[11] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer, "Wrangler: Interactive visual specification of data transformation scripts," in *ACM Human Factors in Computing Systems (CHI)*, 2011. [Online]. Available: http://vis.stanford.edu/papers/wrangler

[12] R. Singh and S. Gulwani, "Learning semantic string transformations from examples," *Proceedings of the VLDB Endowment*, vol. 5, no. 8, pp. 740–751, 2012.

[13] B. Chandramouli, M. Ali, J. Goldstein, B. Sezgin, and B. S. Raman, "Data stream management systems for computational finance," *IEEE Computer*, pp. 45–52, December 2010.

[14] C. Tilmes, Y. Yesha, and M. Halem, "Tracking provenance of earth science data," *Earth Science Informatics*, vol. 3, no. 1-2, pp. 59–65, 2010.

[15] O. Raz, P. Koopman, and M. Shaw, "Semantic anomaly detection in online data sources," in *Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on*. IEEE, 2002, pp. 302–312.

[16] A. Sampson, P. Panchekha, T. Mytkowicz, K. S. McKinley, D. Grossman, and L. Ceze, "Expressing and verifying probabilistic assertions," in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2014, p. 14.

[17] R. Nuzzo, "Scientific method: Statistical errors," *Nature*, vol. 506, no. 7487, February 2014.

[18] S. Nadella, "A data culture for everyone." [Online]. Available: http://blogs.microsoft.com/blog/2014/04/15/a-data-culture-for-everyone/