Software Engineering for Machine Learning: A Case Study

Saleema Amershi
Microsoft Research
Redmond, WA USA
samershi@microsoft.com

Andrew Begel
Microsoft Research
Redmond, WA USA
andrew.begel@microsoft.com

Christian Bird
Microsoft Research
Redmond, WA USA
cbird@microsoft.com

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

Harald Gall
University of Zurich
Zurich, Switzerland
gall@ifi.uzh.ch

Ece Kamar
Microsoft Research
Redmond, WA USA
eckamar@microsoft.com

Nachiappan Nagappan
Microsoft Research
Redmond, WA USA
nachin@microsoft.com

Besmira Nushi
Microsoft Research
Redmond, WA USA
besmira.nushi@microsoft.com

Thomas Zimmermann
Microsoft Research
Redmond, WA USA
tzimmer@microsoft.com

***Abstract*—Recent advances in machine learning have stimulated widespread interest within the Information Technology sector on integrating AI capabilities into software and services. This goal has forced organizations to evolve their development processes. We report on a study that we conducted on observing software teams at Microsoft as they develop AI-based applications. We consider a nine-stage workflow process informed by prior experiences developing AI applications (e.g., search and NLP) and data science tools (e.g. application diagnostics and bug reporting). We found that various Microsoft teams have united this workflow into preexisting, well-evolved, Agile-like software engineering processes, providing insights about several essential engineering challenges that organizations may face in creating large-scale AI solutions for the marketplace. We collected some best practices from Microsoft teams to address these challenges. In addition, we have identified three aspects of the AI domain that make it fundamentally different from prior software application domains: 1) discovering, managing, and versioning the data needed for machine learning applications is much more complex and difficult than other types of software engineering, 2) model customization and model reuse require very different skills than are typically found in software teams, and 3) AI components are more difficult to handle as distinct modules than traditional software components — models may be "entangled" in complex ways and experience non-monotonic error behavior. We believe that the lessons learned by Microsoft teams will be valuable to other organizations.**

***Index Terms*—AI, Software engineering, process, data**

I. Introduction

Personal computing. The Internet. The Web. Mobile computing. Cloud computing. Nary a decade goes by without a disruptive shift in the dominant application domain of the software industry. Each shift brings with it new software engineering goals that spur software organizations to evolve their development practices in order to address the novel aspects of the domain.

The latest trend to hit the software industry is around integrating artificial intelligence (AI) capabilities based on advances in machine learning. AI broadly includes technologies for reasoning, problem solving, planning, and learning, among others. Machine learning refers to statistical modeling techniques that have powered recent excitement in the software and services marketplace. Microsoft product teams have used machine learning to create application suites such as Bing Search or the Cortana virtual assistant, as well as platforms such as Microsoft Translator for real-time translation of text, voice, and video, Cognitive Services for vision, speech, and language understanding for building interactive, conversational agents, and the Azure AI platform to enable customers to build their own machine learning applications [1]. To create these software products, Microsoft has leveraged its preexisting capabilities in AI and developed new areas of expertise across the company.

In this paper, we describe a study in which we learned how various Microsoft software teams build software applications with customer-focused AI features. For that, Microsoft has integrated existing Agile software engineering processes with AI-specific workflows informed by prior experiences in developing early AI and data science applications. In our study, we asked Microsoft employees about how they worked through the growing challenges of daily software development specific to AI, as well as the larger, more essential issues inherent in the development of large-scale AI infrastructure and applications. With teams across the company having differing amounts of work experience in AI, we observed that many issues reported by newer teams dramatically drop in importance as the teams mature, while some remain as essential to the practice of large-scale AI. We have made a first attempt to create a process maturity metric to help teams identify how far they have come on their journeys to building AI applications.

As a key finding of our analyses, we discovered three fundamental differences to building applications and platforms for training and fielding machine-learning models than we have seen in prior application domains. First, machine learning is all about data. The amount of effort and rigor it takes to discover, source, manage, and version data is inherently more complex and different than doing the same with software code. Second, building for customizability and extensibility of models require teams to not only have software engineering skills but almost

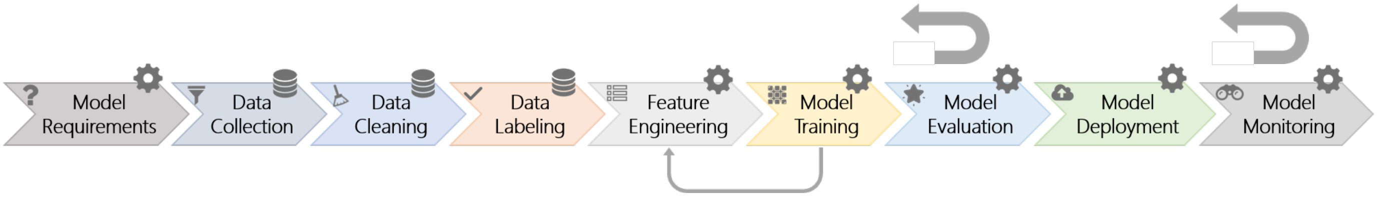

Fig. 1. The nine stages of the machine learning workflow. Some stages are data-oriented (e.g., collection, cleaning, and labeling) and others are model-oriented (e.g., model requirements, feature engineering, training, evaluation, deployment, and monitoring). There are many feedback loops in the workflow. The larger feedback arrows denote that model evaluation and monitoring may loop back to any of the previous stages. The smaller feedback arrow illustrates that model training may loop back to feature engineering (e.g., in representation learning).

always require deep enough knowledge of machine learning to build, evaluate, and tune models from scratch. Third, it can be more difficult to maintain strict module boundaries between machine learning components than for software engineering modules. Machine learning models can be "entangled" in complex ways that cause them to affect one another during training and tuning, even if the software teams building them intended for them to remain isolated from one another.

The lessons we identified via studies of a variety of teams at Microsoft who have adapted their software engineering processes and practices to integrate machine learning can help other software organizations embarking on their own paths towards building AI applications and platforms.

In this paper, we offer the following contributions.

1) A description of how several Microsoft software engineering teams work cast into a nine-stage workflow for integrating machine learning into application and platform development.
2) A set of best practices for building applications and platforms relying on machine learning.
3) A custom machine-learning process maturity model for assessing the progress of software teams towards excellence in building AI applications.
4) A discussion of three fundamental differences in how software engineering applies to machine-learning–centric components vs. previous application domains.

II. Background

A. Software Engineering Processes

The changing application domain trends in the software industry have influenced the evolution of the software processes practiced by teams at Microsoft. For at least a decade and a half, many teams have used feedback-intense Agile methods to develop their software [2], [3], [4] because they needed to be responsive at addressing changing customer needs through faster development cycles. Agile methods have been helpful at supporting further adaptation, for example, the most recent shift to re-organize numerous team's practices around DevOps [5], which better matched the needs of building and supporting cloud computing applications and platforms.[1] The change to DevOps occurred fairly quickly because these teams were able to leverage prior capabilities in continuous integration and diagnostic-gathering, making it simpler to implement continuous delivery.

Process changes not only alter the day-to-day development practices of a team, but also influence the roles that people play. 15 years ago, many teams at Microsoft relied heavily on development triads consisting of a program manager (requirements gathering and scheduling), a developer (programming), and a tester (testing) [6]. These teams' adoption of DevOps combined the roles of developer and tester and integrated the roles of IT, operations, and diagnostics into the mainline software team.

In recent years, teams have increased their abilities to analyze diagnostics-based customer application behavior, prioritize bugs, estimate failure rates, and understand performance regressions through the addition of data scientists [7], [8], who helped pioneer the integration of statistical and machine learning workflows into software development processes. Some software teams employ *polymath* data scientists, who "do it all," but as data science needs to scale up, their roles specialize into domain experts who deeply understand the business problems, modelers who develop predictive models, and platform builders who create the cloud-based infrastructure.

B. ML Workflow

One commonly used machine learning workflow at Microsoft has been depicted in various forms across industry and research [1], [9], [10], [11]. It has commonalities with prior workflows defined in the context of data science and data mining, such as TDSP [12], KDD [13], and CRISP-DM [14]. Despite the minor differences, these representations have in common the data-centered essence of the process and the multiple feedback loops among the different stages. Figure 1 shows a simplified view of the workflow consisting of nine stages.

In the *model requirements* stage, designers decide which features are feasible to implement with machine learning and which can be useful for a given existing product or for a new one. Most importantly, in this stage, they also decide what types of models are most appropriate for the given problem. During *data collection*, teams look for and integrate available datasets (e.g., internal or open source) or collect their own. Often, they might train a partial model using available generic datasets (e.g., ImageNet for object detection), and then use transfer learning together with more specialized data to

[1] https://docs.microsoft.com/en-us/azure/devops/learn/devops-at-microsoft/

train a more specific model (e.g., pedestrian detection). *Data cleaning* involves removing inaccurate or noisy records from the dataset, a common activity to all forms of data science.

Data labeling assigns ground truth labels to each record. For example, an engineer might have a set of images on hand which have not yet been labeled with the objects present in the image. Most of the supervised learning techniques require labels to be able to induce a model. Other techniques (e.g., reinforcement learning) use demonstration data or environment rewards to adjust their policies. Labels can be provided either by engineers themselves, domain experts, or by crowd workers in online crowd-sourcing platforms.

Feature engineering refers to all activities that are performed to extract and select informative features for machine learning models. For some models (e.g. convolutional neural networks), this stage is less explicit and often blended with the next stage, model training. During *model training*, the chosen models (using the selected features) are trained and tuned on the clean, collected data and their respective labels. Then in *model evaluation*, the engineers evaluate the output model on tested or safeguard datasets using pre-defined metrics. For critical domains, this stage might also involve extensive human evaluation. The inference code of the model is then *deployed* on the targeted device(s) and continuously *monitored* for possible errors during real-world execution.

For simplicity the view in Figure 1 is linear, however, machine learning workflows are highly non-linear and contain several feedback loops. For example, if engineers notice that there is a large distribution shift between the training data and the data in the real world, they might want to go back and collect more representative data and rerun the workflow. Similarly, they may revisit their modeling choices made in the first stage, if the problem evolves or if better algorithms are invented. While feedback loops are typical in Agile software processes, the peculiarity of the machine learning workflow is related to the amount of experimentation needed to converge to a good model for the problem. Indeed, the day-to-day work of an engineer doing machine learning involves frequent iterations over the selected model, hyper-parameters, and dataset refinement. Similar experimental properties have been observed in the past in scientific software [15] and hardware/software co-design [16]. This workflow can become even more complex if the system is integrative, containing multiple ML components which interact together in complex and unexpected ways [17].

C. Software Engineering for Machine Learning

The need for adjusting software engineering practices in the recent era has been discussed in the context of hidden technical debt [18] and troubleshooting integrative AI [19], [20]. This work identifies various aspects of ML system architecture and requirements which need to be considered during system design. Some of these aspects include hidden feedback loops, component entanglement and eroded boundaries, non-monotonic error propagation, continuous quality states, and mismatches between the real world and evaluation sets. On a related line of thought, recent work also discusses the impact that the use of ML-based software has on risk and safety concerns of ISO standards [21]. In the last five years, there have been multiple efforts in industry to automate this process by building frameworks and environments to support the ML workflow and its experimental nature [1], [22], [23]. However, ongoing research and surveys show that engineers still struggle to operationalize and standardize working processes [9], [24], [23]. The goal of this work is to uncover detailed insights on ML-specific best practices used by developers at Microsoft. We share these insights with the broader community aspiring that such take-away lessons can be valuable to other companies and engineers.

D. Process Maturity

Software engineers face a constantly changing set of platforms and technologies that they must learn to build the newest applications for the software marketplace. Some engineers learn new methods and techniques in school, and bring them to the organizations they work for. Other learn new skills on the job or on the side, as they anticipate their organization's need for latent talent [25]. Software teams, composed of individual engineers with varying amounts of experience in the skills necessary to professionally build ML components and their support infrastructure, themselves exhibit varying levels of proficiency in their abilities depending on their aggregate experience in the domain.

The software engineering discipline has long considered software process improvement as one of its vital functions. Researchers and practitioners in the field have developed several well-known metrics to assess it, including the Capability Maturity Model (CMM) [26] and Six Sigma [27]. CMM rates the software processes of organizations on five levels, from initial (*ad hoc* processes), repeatable, defined, capable (i.e., quantitatively measured), and efficient (i.e., deliberate process improvement). Inspired by CMM, we build a first maturity model for teams building systems and platforms that integrate machine learning components.

III. Study

We collected data in two phases: an initial set of interviews to gather the major topics relevant to our research questions and a wide-scale survey about the identified topics. Our study design was approved by Microsoft's Ethics Advisory Board.

A. Interviews

Because the work practice around building and integrating machine learning into software and services is still emerging and is not uniform across all product teams, there is no systematic way to identify the key stakeholders on the topic of adoption. We therefore used a snowball sampling strategy, starting with (1) leaders of teams with mature use of machine learning (ML) (e.g., Bing), (2) leaders of teams where AI is a major aspect of the user experience (e.g., Cortana), and (3) people conducting company-wide internal training in AI and ML. As we chose informants, we picked a variety of teams

TABLE I
THE STAKEHOLDERS WE INTERVIEWED FOR THE STUDY.

Id	Role	Product Area	Manager?
I1	Applied Scientist	Search	Yes
I2	Applied Scientist	Search	Yes
I3	Architect	Conversation	Yes
I4	Engineering Manager	Vision	Yes
I5	General Manager	ML Tools	Yes
I6	Program Manager	ML Tools	Yes
I7	Program Manager	Productivity Tools	Yes
I8	Researcher	ML Tools	Yes
I9	Software Engineer	Speech	Yes
I10	Program Manager	AI Platform	No
I11	Program Manager	Community	No
I12	Scientist	Ads	No
I13	Software Engineer	Vision	No
I14	Software Engineer	Vision	No

1. Part 1
   1.1. Background and demographics:
      1.1.1. years of AI experience
      1.1.2. primary AI use case*
      1.1.3. team effectiveness rating
      1.1.4. source of AI components
   1.2. Challenges*
   1.3. Time spent on each of the nine workflow activities
   1.4. Time spent on cross-cutting activities
2. Part 2 (repeated for two activities where most time spent)
   2.1. Tools used*
   2.2. Effectiveness rating
   2.3. Maturity ratings
3. Part 3
   3.1. Dream tools*
   3.2. Best practices*
   3.3. General comments*

Fig. 2. The structure of the study's questionnaire. An asterisk indicates an open-response item.

to get different levels of experience and different parts of the ecosystem (products with AI components, AI frameworks and platforms, AI created for external companies). In all, we interviewed 14 software engineers, largely in senior leadership roles. These are shown in Table I. The interviews were semi-structured and specialized to each informant's role. For example, when interviewing Informant I3, we asked questions related to his work overseeing teams building the product's architectural components.

B. Survey

Based on the results of the interviews, we designed an open-ended questionnaire whose focus was on existing work practice, challenges in that work practice, and best practices (Figure 2). We asked about challenges both directly and indirectly by asking informants to imagine "dream tools" and improvements that would make their work practice better. We sent the questionnaire to 4195 members of internal mailing lists on the topics of AI and ML. 551 software engineers responded, giving us a 13.6% response rate. For each open-response item, between two and four researchers analyzed the responses through a card sort. Then, the entire team reviewed the card sort results for clarity and consistency.

Respondents were fairly well spread across all divisions of the company and came from a variety of job roles: Data and applied science (42%), Software engineering (32%), Program management (17%), Research (7%), and other (1%). 21% of respondents were managers and 79% were individual contributors, helping us balance out the majority manager perspective in our interviews.

In the next sections, we discuss our interview and survey results, starting with the range of AI applications developed by Microsoft, diving into best practices that Microsoft engineers have developed to address some of the essential challenges in building large-scale AI applications and platforms, showing how the perception of the importance of the challenges changes as teams gain experience building AI applications, and finally, describing our proposed AI process maturity model.

IV. APPLICATIONS OF AI

Many teams across Microsoft have augmented their applications with machine learning and inference, some in some surprising domains. We asked survey respondents for the ways that they used AI on their teams. We card sorted this data twice, once to capture the application domain in which AI was being applied, and a second time to look at the (mainly) ML algorithms used to build that application.

We found AI is used in traditional areas such as search, advertising, machine translation, predicting customer purchases, voice recognition, and image recognition, but also saw it being used in novel areas, such as identifying customer leads, providing design advice for presentations and word processing documents, providing unique drawing features, healthcare, and improving gameplay. In addition, machine learning is being used heavily in infrastructure projects to manage incident reporting, identify the most likely causes for bugs, monitor fraudulent fiscal activity, and to monitor network streams for security breaches.

Respondents used a broad spectrum of ML approaches to build their applications, from classification, clustering, dynamic programming, and statistics, to user behavior modeling, social networking analysis, and collaborative filtering. Some areas of the company specialized further, for instance, Search worked heavily with ranking and relevance algorithms along with query understanding. Many divisions in the company work on natural language processing, developing tools for entity recognition, sentiment analysis, intent prediction, summarization, machine translation, ontology construction, text similarity, and connecting answers to questions. Finance and Sales have been keen to build risk prediction models and do forecasting. Internal resourcing organizations make use of decision optimization algorithms such as resource optimization, planning, pricing, bidding, and process optimization.

The takeaway for us was that integration of machine learning components is happening all over the company, not just on teams historically known for it. Thus, we could tell that we were not just hearing from one niche corner of the company, but in fact, we received responses from a broad range of perspectives spread throughout.

V. Best Practices with Machine Learning in Software Engineering

In this section, we present our respondents' viewpoints on some of the essential challenges associated with building large-scale ML applications and platforms and how they address them in their products. We categorized the challenges by card sorting interview and survey free response questions, and then used our own judgment as software engineering and AI researchers to highlight those that are essential to the practice of AI on software teams.

A. End-to-end pipeline support

As machine learning components have become more mature and integrated into larger software systems, our participants recognized the importance of integrating ML development support into the traditional software development infrastructure. They noted that having a seamless development experience covering (possibly) all the different stages described in Figure 1 was important to automation. However, achieving this level of integration can be challenging because of the different characteristics of ML modules compared with traditional software components. For example, previous work in this field [18], [19] found that variation in the inherent uncertainty (and error) of data-driven learning algorithms and complex component entanglement caused by hidden feedback loops could impose substantial changes (even in specific stages) which were previously well understood in software engineering (e.g., specification, testing, debugging, to name a few). Nevertheless, due to the experimental and even more iterative nature of ML development, unifying and automating the day-to-day workflow of software engineers reduces overhead and facilitate progress in the field.

Respondents report to leverage internal infrastructure in the company (e.g. AEther[2]) or they have built pipelines specialized to their own use cases. It is important to develop a "rock solid, data pipeline, capable of continuously loading and massaging data, enabling engineers to try out many permutations of AI algorithms with different hyper-parameters without hassle." The pipelines created by these teams are automated, supporting training, deployment, and integration of models with the product they are a part of. In addition, some pipeline engineers indicated that "rich dashboards" showing the value provided to users are useful.

Several respondents develop openly available IDEs to enable Microsoft's customers to build and deploy their models (e.g. Azure ML for Visual Studio Code[3] and Azure ML Studio[4]). According to two of our interviewees, the goal of these environments is to help engineers discover, gather, ingest, understand, and transform data, and then train, deploy, and maintain models. In addition, these teams customize the environments to make them easier to use by engineers with varying levels of experience. "Visual tools help beginning data scientists when getting started, but once they know the ropes and branch out, such tools may get in their way and they may need something else."

B. Data availability, collection, cleaning, and management

Since many machine learning techniques are centered around learning from large datasets, the success of ML-centric projects often heavily depends on data availability, quality and management [28]. Labeling datasets is costly and time-consuming, so it is important to make them available for use within the company (subject to compliance constraints). Our respondents confirm that it is important to "reuse the data as much as possible to reduce duplicated effort." In addition to availability, our respondents focus most heavily on supporting the following data attributes: "accessibility, accuracy, authoritativeness, freshness, latency, structuredness, ontological typing, connectedness, and semantic joinability." Automation is a vital cross-cutting concern, enabling teams to more efficiently aggregate data, extract features, synthesize labelled examples. The increased efficiency enables teams to "speed up experimentation and work with live data while they experiment with new models."

We found that Microsoft teams have found it necessary to blend data management tools with their ML frameworks to avoid the fragmentation of data and model management activities. A fundamental aspect of data management for machine learning is the rapid evolution of data sources. Continuous changes in data may originate either from (i) operations initiated by engineers themselves, or from (ii) incoming fresh data (e.g., sensor data, user interactions). Either case requires rigorous data versioning and sharing techniques, for example, "Each model is tagged with a provenance tag that explains with which data it has been trained on **and** which version of the model. Each dataset is tagged with information about where it originated from and which version of the code was used to extract it (and any related features)." This practice is used for mapping datasets to deployed models or for facilitating data sharing and reusability.

C. Education and Training

The integration of machine learning continues to become more ubiquitous in customer-facing products, for example, machine learning components are now widely used in productivity software (e.g., email, word processing) and embedded devices (i.e., edge computing). Thus, engineers with traditional software engineering backgrounds need to learn how to work alongside of the ML specialists. A variety of players within Microsoft have found it incredibly valuable to scaffold their engineers' education in a number of ways. First, the company hosts a twice-yearly internal conference on machine learning and data science, with at least one day devoted to introductions

[2] https://www.slideshare.net/MSTechCommunity/ai-microsoft-how-we-do-it-and-how-you-can-too

[3] https://marketplace.visualstudio.com/items?itemName=ms-toolsai.vscode-ai

[4] https://azure.microsoft.com/en-us/services/machine-learning-studio/

to the basics of technologies, algorithms, and best practices. In addition, employees give talks about internal tools and the engineering details behind novel projects and product features, and researchers present cutting-edge advances they have seen and contributed to academic conferences. Second, a number of Microsoft teams host weekly open forums on machine learning and deep learning, enabling practitioners to get together and learn more about AI. Finally, mailing lists and online forums with thousands of participants enable anyone to ask and answer technical and pragmatic questions about AI and machine learning, as well as frequently share recent results from academic conferences.

D. Model Debugging and Interpretability

Debugging activities for components that learn from data not only focus on programming bugs, but also focus on inherent issues that arise from model errors and uncertainty. Understanding when and how models fail to make accurate predictions is an active research area [29], [30], [31], which is attracting more attention as ML algorithms and optimization techniques become more complex. Several survey respondents and the larger Explainable AI community [32], [33] propose to use more interpretable models, or to develop visualization techniques that make black-box models more interpretable. For larger, multi-model systems, respondents apply modularization in a conventional, layered, and tiered software architecture to simplify error analysis and debuggability.

E. Model Evolution, Evaluation, and Deployment

ML-centric software goes through frequent revisions initiated by model changes, parameter tuning, and data updates, the combination of which has a significant impact on system performance. A number of teams have found it important to employ rigorous and agile techniques to evaluate their experiments. They developed systematic processes by adopting combo-flighting techniques (i.e., flighting a combination of changes and updates), including multiple metrics in their experiment score cards, and performing human-driven evaluation for more sensitive data categories. One respondent's team uses "score cards for the evaluation of flights and storing flight information: How long has it been flighted, metrics for the flight, etc." Automating tests is as important in machine learning as it is in software engineering; teams create carefully put-together test sets that capture what their models should do. However, it is important that a human remains in the loop. One respondent said, "we spot check and have a human look at the errors to see why this particular category is not doing well, and then hypothesize to figure out problem source."

Fast-paced model iterations require more frequent deployment. To ensure that system deployment goes smoothly, several engineers recommend not only to automate the training and deployment pipeline, but also to integrate model building with the rest of the software, use common versioning repositories for both ML and non-ML codebases, and tightly couple the ML and non-ML development sprints and standups.

F. Compliance

Microsoft issued a set of principles around uses of AI in the open world. These include fairness, accountability, transparency, and ethics. All teams at Microsoft have been asked to align their engineering practices and the behaviors of fielded software and services in accordance with these principles. Respect for them is a high priority in software engineering and AI and ML processes and practices. A discussion of these concerns is beyond the scope of this paper. To learn more about Microsoft's commitments to this important topic, please read about its approach to AI.[5]

G. Varied Perceptions

We found that as a number of product teams at Microsoft integrated machine learning components into their applications, their ability to do so effectively was mediated by the amount of prior experience with machine learning and data science. Some teams fielded data scientists and researchers with decades of experience, while others had to grow quickly, picking up their own experience and more-experienced team members on the way. Due to this heterogeneity, we expected that our survey respondents' perceptions of the challenges their teams' faced in practicing machine learning would vary accordingly.

We grouped the respondents into three buckets (low, medium, and high), evenly divided by the number of years of experience respondents personally had with AI. First, we ranked each of the card sorted categories of respondents' challenges divided by the AI experience buckets. This list is presented in Table II, initially sorted by the respondents with low experience with AI.

Two things are worth noticing. First, across the board, Data Availability, Collection, Cleaning, and Management, is ranked as the top challenge by many respondents, no matter their experience level. We find similarly consistent ranking for issues around the categories of end-to-end pipeline support and collaboration and working culture. Second, some of the challenges rise or fall in importance as the respondents' experience with AI differs. For example, education and training is far more important to those with low experience levels in AI than those with more experience. In addition, respondents with low experience rank challenges with integrating AI into larger systems higher than those with medium or high experience. This means that as individuals (and their teams) gain experience building applications and platforms that integrate ML, their increasing skills help shrink the importance of some of the challenges they perceive. Note, the converse also occurs. Challenges around tooling, scale, and model evolution, evaluation, and deployment are more important for engineers with a lot of experience with AI. This is very likely because these more experienced individuals are tasked with the more essentially difficult engineering tasks on their team; those with low experience are probably tasked to easier problems until they build up their experience.

[5] https://www.microsoft.com/en-us/ai/our-approach-to-ai

TABLE II
THE TOP-RANKED CHALLENGES AND PERSONAL EXPERIENCE WITH AI. RESPONDENTS WERE GROUPED INTO THREE BUCKETS (LOW, MEDIUM, HIGH) BASED ON THE 33RD AND 67TH PERCENTILE OF THE NUMBER OF YEARS OF AI EXPERIENCE THEY PERSONALLY HAD (N=308). THE COLUMN *Frequency* SHOWS THE INCREASE/DECREASE OF THE FREQUENCY IN THE MEDIUM AND HIGH BUCKETS COMPARED TO THE LOW BUCKETS. THE COLUMN *Rank* SHOWS THE RANKING OF THE CHALLENGES WITHIN EACH EXPERIENCE BUCKET, WITH 1 BEING THE MOST FREQUENT CHALLENGE.

	Frequency			Rank		
	Medium	High			Experience	
Challenge	vs. Low	vs. Low	Trend	Low	Medium	High
Data Availability, Collection, Cleaning, and Management	-2%	60%		1	1	1
Education and Training	-69%	-78%		1	5	9
Hardware Resources	-32%	13%		3	8	6
End-to-end pipeline support	65%	41%		4	2	4
Collaboration and working culture	19%	69%		5	6	6
Specification	2%	50%		5	8	8
Integrating AI into larger systems	-49%	-62%		5	16	13
Education: Guidance and Mentoring	-83%	-81%		5	21	18
AI Tools	144%	193%		9	3	2
Scale	154%	210%		10	4	3
Model Evolution, Evaluation, and Deployment	137%	276%		15	6	4

We also compared the overall frequency of each kind of challenge using the same three buckets of AI experience. Looking again at the top ranked challenge, Data Availability, Collection, Cleaning, and Management, we notice that it was reported by low and medium experienced respondents at similar rates, but represented a lot more of the responses (60%) given by those with high experience. This also happened for challenges related to Specifications. However, when looking at Education and Training, Integrating AI into larger systems, and Education: Guidance and Mentoring, their frequency drops significantly from the rate reported by the low experience bucket than reported by the medium and high buckets. We interpret this to mean that these challenges were less important to the medium and high experience respondents than to those with low experience levels. Thus, this table gives a big picture of both which problems are perceived as most important *within* each experience bucket, and which problems are perceived as most important *across* the buckets.

Finally, we conducted a logistic regression analysis to build a model that could explain the differences in frequency when controlling for personal AI experience, team AI experience, overall work experience, the number of concurrent AI projects, and whether or not a respondent had formal education in machine learning or data science techniques. We found five significant coefficients:

- Education and Training was negatively correlated with personal AI experience with a coefficient of -0.18 ($p < 0.02$), meaning that people with less AI experience found this to be a more important issue.
- Educating Others was positively correlated with personal AI experience with a coefficient of 0.26 ($p < 0.01$), meaning that people with greater AI experience found this to be a more important issue.
- Tool issues are positively correlated with team AI experience with a coefficient of 0.13 ($p < 0.001$), meaning that as the team gains experience working on AI projects, the degree to which they rely on others' and their own tools goes up, making them think about their impact more often.
- End-to-end pipeline support was positively correlated with formal education ($p < 0.01$), implying that only those with formal education were working on building such a pipeline.
- Specifications were also positively correlated with formal education ($p < 0.03$), implying that those with formal education are the ones who write down the specifications for their models and engineering systems.

The lesson we learn from these analyses is that the kinds of issues that engineers perceive as important change as they grow in their experience with AI. Some concerns are transitory, related to one's position within the team and the accidental complexity of working together. Several others are more fundamental to the practice of integrating machine learning into software applications, affecting many engineers, no matter their experience levels. Since machine learning-based applications are expected to continue to grow in popularity, we call for further research to address these important issues.

VI. TOWARDS A MODEL OF ML PROCESS MATURITY

As we saw in Section V-G, we see some variance in the experience levels of AI in software teams. That variation affects their perception of the engineering challenges to be addressed in their day-to-day practices. As software teams mature and gel, they can become more effective and efficient in delivering machine learning-based products and platforms.

To capture the maturity of ML more precisely than using a simple years-of-experience number, we created a maturity model with six dimensions evaluating whether each workflow stage: (1) has defined goals, (2) is consistently implemented, (3) documented, (4) automated, (5) measured and tracked, and (6) continuously improved. The factors are loosely based on the concepts behind the Capability Maturity Model (CMM) [26] and Six Sigma [27], which are widely used in software development to assess and improve maturity of software projects.

In the survey, we asked respondents to report the maturity for the two workflow stages that each participant spent the most time on (measured by number of hours they reported spending on each activity). Specifically, we asked participants to rate their agreement with the following statements $S_1..S_6$ (bold text was in the original survey) using a Likert response format from *Strongly Disagree* (1) to *Strongly Agree* (5):

S1: My team has **goals defined** for what to accomplish with this activity.
S2: My team does this activity in a **consistent manner**.
S3: My team has largely **documented** the practices related to this activity.
S4: My team does this activity mostly in an **automated way**.
S5: My team **measures and tracks** how effective we are at completing this activity.
S6: My team **continuously improves** our practices related to this activity.

We gathered this data for the stages that respondents were most familiar with because we found that they often specialize in various stages of the workflow. This question was intended to be lightweight so that respondents could answer easily, while at the same time accounting for the wide variety of ML techniques applied. Rather than being prescriptive (i.e., do this to get to the next maturity level), our intention was to be descriptive (e.g., how much automation is there in a particular workflow stage? how well is a workflow stage documented?). More work is needed to define maturity levels similar to CMM.

To analyze the responses, we defined an *Activity Maturity Index* (AMI) to combine the individual scores into a single measure. This index is the average of the agreement with the six maturity statements $S_1..S_6$. As a means of validating the Maturity Index, we asked participants to rate the *Activity Effectiveness* (AE) by answering "How effective do you think your team's practices around this activity are on a scale from 1 (poor) to 5 (excellent)?". The Spearman correlation between the Maturity Index and the Effectiveness was between 0.4982 and 0.7627 (all statistically significant at $p < 0.001$) for all AI activities. This suggests that the Maturity Index is a valid composite measure that can capture the maturity and effectiveness of AI activities.

In addition to the Activity Maturity Index and Activity Effectiveness, we collected an *Overall Effectiveness* (OE) score by asking respondents the question "How effectively does your team work with AI on a scale from 1 (poor) to 5 (excellent)" Having the AMI, AE, and OE measures allowed us to compare the maturity and effectiveness of different organizations, disciplines, and application domains within Microsoft, and identify areas for improvement. We plot one of these comparisons in Figure 3 and show the average overall effectiveness scores divided by nine of the most represented AI application domains in our survey. There are two things to notice. First, the spread of the y-values indicates that the OE metric can numerically distinguish between teams, meaning that some respondents feel their teams are at different levels of maturity than others. Second, an ANOVA and Scott Knott test show significant differences in the reported values, demonstrating the potential value of this metric to identify the various ML process maturity levels.

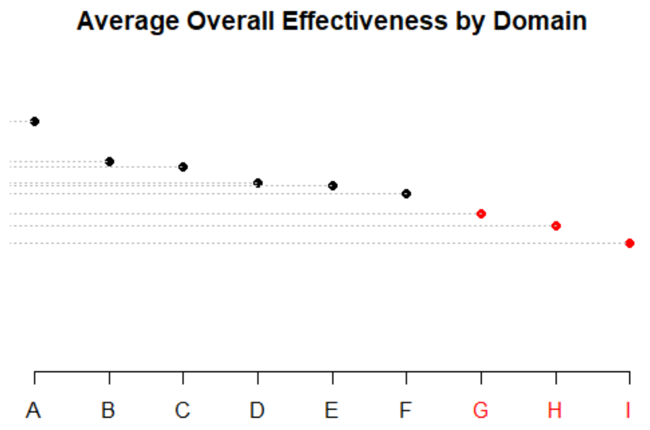

Fig. 3. The average overall effectiveness (OE) of a team's ML practices divided by application domain (anonymized). The y-axis labels have been elided for confidentiality. An ANOVA and Scott Knott test identified two distinct groups to the OE metric, labeled in black (A–F) and red (G–I).

We recognize that these metrics represent a first attempt at quantifying a process metric to enable teams to assess how well they practice ML. In future work, we will refine our instrument and further validate its utility.

VII. Discussion

In this section, we synthesize our findings into three observations of some fundamental differences in the way that software engineering has been adapted to support past popular application domains and how it can be adapted to support artificial intelligence applications and platforms. There may be more differences, but from our data and discussions with ML experts around Microsoft, these three rose to prominence.

A. Data discovery and management

Just as software engineering is primarily about the code that forms shipping software, ML is all about the data that powers learning models. Software engineers prefer to design and build systems which are elegant, abstract, modular, and simple. By contrast, the data used in machine learning are voluminous, context-specific, heterogeneous, and often complex to describe. These differences result in difficult problems when ML models are integrated into software systems at scale.

Engineers have to find, collect, curate, clean, and process data for use in model training and tuning. All the data has to be stored, tracked, and versioned. While software APIs are described by specifications, datasets rarely have explicit schema definitions to describe the columns and characterize their statistical distributions. However, due to the rapid iteration involved in ML, the data schema (and the data) change frequently, even many times per day. When data is ingested from large-scale diagnostic data feeds, if ML engineers want to change which data values are collected, they must wait for the engineering systems to be updated, deployed, and propagated before new data can arrive. Even "simple" changes can have significant impacts on the volume of data collected, potentially impacting applications through altered performance characteristics or increased network bandwidth usage.

While there are very well-designed technologies to version code, the same is not true for data. A given data set may contain data from several different schema regimes. When a single engineer gathers and processes this data, they can keep track of these unwritten details, but when project sizes scale, maintaining this tribal knowledge can become a burden. To help codify this information into a machine-readable form, Gebru *et al.* propose to use data sheets inspired by electronics to more transparently and reliably track the metadata characteristics of these datasets [34]. To compare datasets against each other, the Datadiff [35] tool enables developers to formulate viable transformation functions over data samples.

B. Customization and Reuse

While it is well-understood how much work it takes to customize and reuse code components, customizing ML models can require much more. In software, the primary units of reuse are functions, algorithms, libraries, and modules. A software engineer can find the source code for a library (e.g. on Github), fork it, and easily make changes to the code, using the same skills they use to develop their own software.

Although fully-trained ML models appear to be functions that one can call for a given input, the reality is far more complex. One part of a model is the algorithm that powers the particular machine learning technique being used (e.g., SVM or neural nets). Another is the set of parameters that controls the function (e.g., the SVM support vectors or neural net weights) and are learned during training. If an engineer wants to apply the model on a similar domain as the data it was originally trained on, reusing it is straightforward. However, more signficant changes are needed when one needs to run the model on a different domain or use a slightly different input format. One cannot simply change the parameters with a text editor. In fact, the model may require retraining, or worse, may need to be replaced with another model. Both require the software developer to have machine learning skills, which they may never have learned. Beyond that, retraining or rebuilding the model requires additional training data to be discovered, collected, and cleaned, which can take as much work and expertise as the original model's authors put in.

C. ML Modularity

Another key attribute of engineering large-scale software systems is modularity. Modules are separated and isolated to ensure that developing one component does not interfere with the behavior of others under development. In addition, software modularity is strengthened by Conway's Law, which makes the observation that the teams that build each component of the software organize themselves similarly to its architecture. Thus, separate modules are often assigned to separate teams. Module interactions are controlled by APIs which do dual duty to enable software modules to remain apart, but also describe the interfaces to minimize the amount of communication needed between separate teams to make their modules work together [36], [37].

Maintaining strict module boundaries between machine learned models is difficult for two reasons. First, models are not easily extensible. For example, one cannot (yet) take an NLP model of English and add a separate NLP model for ordering pizza and expect them to work properly together. Similarly, one cannot take that same model for pizza and pair it with an equivalent NLP model for French and have it work. The models would have to be developed and trained together.

Second, models interact in non-obvious ways. In large-scale systems with more than a single model, each model's results will affect one another's training and tuning processes. In fact, one model's effectiveness will change as a result of the other model, even if their code is kept separated. Thus, even if separate teams built each model, they would have to collaborate closely in order to properly train or maintain the full system. This phenomenon (also referred to as *component entanglement*) can lead to non-monotonic error propagation, meaning that improvements in one part of the system might decrease the overall system quality because the rest of the system is not tuned to the latest improvements. This issue is even more evident in cases when machine learning models are not updated in a compatible way and introduce new, previously unseen mistakes that break the interaction with other parts of the system which rely on it.

VIII. Limitations

Our case study was conducted with teams at Microsoft, a large, world-wide software company with a diverse portfolio of software products. It is also one of the largest purveyors of machine learning-based products and platforms. Some findings are likely to be specific to the Microsoft teams and team members who participated in our interviews and surveys. Nevertheless, given the high variety of projects represented by our informants, we expect that many of the lessons we present in this paper will apply to other companies. Some of our findings depend on the particular ML workflow used by some software teams at Microsoft. The reader should be able to identify how our model abstractions fit into the particulars of the models they use. Finally, interviews and surveys rely on self-selected informants and self-reported data. Wherever appropriate, we stated that findings were our informants' perceptions and opinions. This is especially true with this implementation of our ML process maturity model, which triangulated its measures against other equally subjective measures with no objective baseline. Future implementations of the maturity model should endeavor to gather objective measures of team process performance and evolution.

IX. Conclusion

Many teams at Microsoft have put significant effort into developing an extensive portfolio of AI applications and platforms by integrating machine learning into existing software engineering processes and cultivating and growing ML talent. In this paper, we described the results of a study to learn more about the process and practice changes undertaken by a number of Microsoft teams in recent years. From these findings, we synthesized a set of best practices to address issues fundamental to the large-scale development and deployment of ML-based applications. Some reported issues were correlated with the respondents' experience with AI, while others were applicable to most respondents building AI applications. We presented a ML process maturity metric to help teams self-assess how well they work with machine learning and offer guidance towards improvements. Finally, we identified three aspects of the AI domain that make it fundamentally different than prior application domains. Their impact will require significant research efforts to address in the future.

References

[1] M. Salvaris, D. Dean, and W. H. Tok, "Microsoft AI Platform," in *Deep Learning with Azure*. Springer, 2018, pp. 79–98.

[2] A. Begel and N. Nagappan, "Usage and perceptions of agile software development in an industrial context: An exploratory study," in *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, Sept 2007, pp. 255–264.

[3] A. Begel and N. Nagappan, "Pair programming: What's in it for me?" in *Proc. of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 2008, pp. 120–128.

[4] B. Murphy, C. Bird, T. Zimmermann, L. Williams, N. Nagappan, and A. Begel, "Have agile techniques been the silver bullet for software development at microsoft?" in *2013 ACM/IEEE Intl. Symp. on Empirical Software Engineering and Measurement*, Oct 2013, pp. 75–84.

[5] M. Senapathi, J. Buchan, and H. Osman, "DevOps capabilities, practices, and challenges: Insights from a case study," in *Proc. of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, 2018, pp. 57–67.

[6] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining mental models: A study of developer work habits," in *Proc. of the 28th International Conference on Software Engineering*, 2006, pp. 492–501.

[7] M. Kim, T. Zimmermann, R. DeLine, and A. Begel, "The emerging role of data scientists on software development teams," in *Proc. of the 38th International Conference on Software Engineering*, 2016, pp. 96–107.

[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*, vol. 44, no. 11, pp. 1024–1038, 2018.

[9] C. Hill, R. Bellamy, T. Erickson, and M. Burnett, "Trials and tribulations of developers of intelligent systems: A field study," in *Visual Languages and Human-Centric Computing (VL/HCC), 2016 IEEE Symposium on*. IEEE, 2016, pp. 162–170.

[10] "Machine learning workflow," https://cloud.google.com/ml-engine/docs/tensorflow/ml-solutions-overview, accessed: 2018-09-24.

[11] K. Patel, J. Fogarty, J. A. Landay, and B. Harrison, "Investigating statistical machine learning as a tool for software development," in *Proc. of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2008, pp. 667–676.

[12] "The Team Data Science Process," https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/, accessed: 2018-09-24.

[13] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "The KDD process for extracting useful knowledge from volumes of data," *Communications of the ACM*, vol. 39, no. 11, pp. 27–34, 1996.

[14] R. Wirth and J. Hipp, "CRISP-DM: Towards a standard process model for data mining," in *Proc. 4th Intl. Conference on Practical Applications of Knowledge Discovery and Data mining*, 2000, pp. 29–39.

[15] J. E. Hannay, C. MacLeod, J. Singer, H. P. Langtangen, D. Pfahl, and G. Wilson, "How do scientists develop and use scientific software?" in *Proc. of the 2009 ICSE workshop on Software Engineering for Computational Science and Engineering*. IEEE Computer Society, 2009, pp. 1–8.

[16] G. De Michell and R. K. Gupta, "Hardware/software co-design," *Proc. of the IEEE*, vol. 85, no. 3, pp. 349–365, 1997.

[17] D. Bohus, S. Andrist, and M. Jalobeanu, "Rapid development of multimodal interactive systems: a demonstration of platform for situated intelligence," in *Proc. of the 19th ACM International Conference on Multimodal Interaction*. ACM, 2017, pp. 493–494.

[18] 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 *NIPS*, 2015.

[19] B. Nushi, E. Kamar, E. Horvitz, and D. Kossmann, "On human intellect and machine failures: Troubleshooting integrative machine learning systems." in *AAAI*, 2017, pp. 1017–1025.

[20] S. Andrist, D. Bohus, E. Kamar, and E. Horvitz, "What went wrong and why? diagnosing situated interaction failures in the wild," in *ICSR*. Springer, 2017, pp. 293–303.

[21] R. Salay, R. Queiroz, and K. Czarnecki, "An analysis of ISO 26262: Using machine learning safely in automotive software," *arXiv preprint arXiv:1709.02435*, 2017.

[22] D. Baylor, E. Breck, H.-T. Cheng, N. Fiedel, C. Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc *et al.*, "TFX: A tensorflow-based production-scale machine learning platform," in *Proc. of the 23rd ACM SIGKDD*. ACM, 2017, pp. 1387–1395.

[23] V. Sridhar, S. Subramanian, D. Arteaga, S. Sundararaman, D. Roselli, and N. Talagala, "Model governance: Reducing the anarchy of production ML," in *USENIX*. USENIX Association, 2018, pp. 351–358.

[24] T. Wuest, D. Weimer, C. Irgens, and K.-D. Thoben, "Machine learning in manufacturing: advantages, challenges, and applications," *Production & Manufacturing Research*, vol. 4, no. 1, pp. 23–45, 2016.

[25] J. Sillito and A. Begel, "App-directed learning: An exploratory study," in *6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, May 2013, pp. 81–84.

[26] C. Weber, B. Curtis, and M. B. Chrissis, *The capability maturity model, guidelines for improving the software process*. Harlow: Addison Wesley, 1994.

[27] M. Alexander, *Six Sigma: The breakthrough management strategy revolutionizing the world's top corporations*. Taylor & Francis, 2001.

[28] N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich, "Data management challenges in production machine learning," in *Proc. of the 2017 ACM SIGMOD*, 2017, pp. 1723–1726.

[29] T. Kulesza, M. Burnett, W.-K. Wong, and S. Stumpf, "Principles of explanatory debugging to personalize interactive machine learning," in *Proc. of the 20th International Conference on Intelligent User Interfaces*, 2015, pp. 126–137.

[30] S. Amershi, M. Chickering, S. M. Drucker, B. Lee, P. Simard, and J. Suh, "Modeltracker: Redesigning performance analysis tools for machine learning," in *Proc. of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, 2015, pp. 337–346.

[31] B. Nushi, E. Kamar, and E. Horvitz, "Towards accountable AI: Hybrid human-machine analyses for characterizing system failure," in *HCOMP*, 2018, pp. 126–135.

[32] D. Gunning, "Explainable artificial intelligence (XAI)," *Defense Advanced Research Projects Agency (DARPA)*, 2017.

[33] D. S. Weld and G. Bansal, "Intelligible artificial intelligence," *arXiv preprint arXiv:1803.04263*, 2018.

[34] T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. M. Wallach, H. D. III, and K. Crawford, "Datasheets for datasets," *CoRR*, vol. abs/1803.09010, 2018.

[35] C. Sutton, T. Hobson, J. Geddes, and R. Caruana, "Data diff: Interpretable, executable summaries of changes in distributions for data wrangling," in *Proc. of the 24th ACM SIGKDD*. ACM, 2018, pp. 2279–2288.

[36] C. R. B. de Souza, D. Redmiles, and P. Dourish, "“Breaking the Code”, moving between private and public work in collaborative software development," in *Proc. of the 2003 International ACM SIGGROUP Conference on Supporting Group Work*, 2003, pp. 105–114.

[37] C. R. B. de Souza, D. Redmiles, L.-T. Cheng, D. Millen, and J. Patterson, "Sometimes you need to see through walls: A field study of application programming interfaces," in *Proc. of the 2004 ACM Conference on Computer Supported Cooperative Work*, 2004, pp. 63–71.