Sensemaking Practices in the Everyday Work of AI/ML Software Engineering

Christine T. Wolf
IBM Research – Almaden
San Jose, CA, USA
ctwolf@us.ibm.com

Drew Paine
Lawrence Berkeley National Laboratory
Berkeley, CA, USA
pained@lbl.gov

ABSTRACT

This paper considers *sensemaking* as it relates to everyday software engineering (SE) work practices and draws on a multi-year ethnographic study of SE projects at a large, global technology company building digital services infused with artificial intelligence (AI) and machine learning (ML) capabilities. Our findings highlight the breadth of sensemaking practices in AI/ML projects, noting developers' efforts to make sense of AI/ML environments (e.g., algorithms/methods and libraries), of AI/ML model ecosystems (e.g., pre-trained models and "upstream" models), and of business-AI relations (e.g., how the AI/ML service relates to the domain context and business problem at hand). This paper builds on recent scholarship drawing attention to the integral role of sensemaking in everyday SE practices by empirically investigating how and in what ways AI/ML projects present software teams with emergent sensemaking requirements and opportunities.

CCS CONCEPTS

• Software and its engineering • Computing methodologies ~Artificial intelligence • Applied computing ~Enterprise computing • Human-centered computing ~Collaborative and social computing ~Empirical studies in collaborative and social computing

KEYWORDS

Sensemaking, AI/ML, Software Engineering, Process Theories, Enterprise computing, Ethnography

ACM Reference format:
Christine T. Wolf and Drew Paine. 2020. Sensemaking Practices in the Everyday Work of AI/ML Software Engineering. In *IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20). ACM, New York, NY, USA, 7 pages.* https://doi.org/10.1145/3387940.3391496

https://doi.org/10.1145/3387940.3391496

1 Introduction

Process theories of software engineering (SE) aim to understand how SE work unfolds in practice and the implications of these dynamics on the nature, quality, and experience of software systems [19]. Ralph [16] describes process theories as "a system of ideas intended to explain (and possibly to describe, to predict, or to analyze) how an entity changes and develops." Ralph's [17] "sensemaking-coevolution-implementation" process theory emphasizes the effort software teams must undertake to manage ambiguity in SE work, describing these efforts as practices that iteratively move between creating coherence of an ambiguous context, refining views of the context and design space, and producing technical artifacts as a result of their understanding of this design space. In this view, sensemaking is broader than the Requirements Engineering (RE) stage of many software process models, instead comprising a core and enduring facet of SE work throughout a project's lifecycle. Indeed, others have also pointed out the central role of sensemaking in SE work, describing SE broadly as a form of "managed sensemaking" [7].

What is sensemaking? Organizational scholar Weick describes sensemaking processes as activities where people make sense of a situation; this is an iterative process where humans gather information and refine their understanding of a situation, adapting their mental models and cognitive framings as they go [24]. Sensemaking is often provoked when surprising or ambiguous situations occur; sensemaking involves "comprehending, redressing surprise, constructing meaning, interacting in pursuit of mutual understanding, and patterning" (p.6). Sensemaking has been a topic of interest to SE researchers for a number of years and has been explored in a variety of contexts – for example, how software teams make sense of RE activities [11], backlogs that represent the work to be done on a product [20], or the products they ultimately deploy into production [3, 8]. End-users also engage in sensemaking as they interact with and debug software programs in production [9] or navigate dense and changing software ecosystems [26]. Sensemaking is both an individual and collective activity [24] and in the case of distributed software teams, it can be a collaborative activity that takes place entirely virtually [22].

Essential for any SE project is the ability of the software team to appropriately identify and comprehend the complex, heterogenous contexts shaping their work [5]. In many conceptualizations of software engineering, RE is seen as the phase in the SE lifecycle where sensemaking primarily occurs [18]. Sensemaking is not a bounded activity though, and occurs continually throughout the SE lifecycle as software teams work to reduce emergent ambiguity and make progress on project goals [17].

In this paper, we engage with the topic of sensemaking in everyday SE work and focus on projects building artificial intelligence (AI) and machine learning (ML) services. AI/ML projects present unique challenges to SE teams, provoking reflection and investigation. What is it about AI/ML projects that makes them particularly challenging for SE teams? Chief among the challenges in AI/ML projects are the dynamism and unpredictability inherent in AI/ML work (e.g., accuracy and reproducibility of AI/ML model outputs cannot be known beforehand), comprehensibility of complex model functions and behavior (e.g., AI explainability), and managing expectations with clients (e.g., idealistic or hyped expectations of what is possible) [2, 10, 12].

The empirical study of SE practices in AI/ML projects is an emergent topic, with nascent work in this area investigating the challenges AI/ML presents for software teams via surveys [1, 10], systematic literature reviews [23], and case studies [14]. We contribute to these discourses by drawing on a multi-year ethnographic study of SE projects at a large, global technology company building digital services infused with AI/ML capabilities. Rather than a focus on challenges *per se*, our focus is on the role of sensemaking in AI/ML SE projects. We ask: *what sensemaking practices emerge in AI/ML software development projects?*

Our paper is laid out as follows. In Section 2, we describe our setting and methods. In Section 3, we outline our key findings. We conclude the paper in Section 4 with a discussion and provocation for future work.

2 Setting and Methods

This paper draws from ongoing ethnographic fieldwork at a large, global technology and services company headquartered in North America (referred to as "TechCorp" or "the company"). Ethnographic studies of software engineering capture *in situ* practices, providing insights into *how* situated action unfolds, but also *why*; such insights expand our scholarly understanding of the nature of SE work in practice and can also help to inform the design of sustainable and humane interventions to improve these forms of work [21]. The overarching focus of this long-term study is understanding the "work of AI" - that is, the various forms of everyday human work and labor practices involved in applied, enterprise AI projects [25]. This focus is exploratory and purposively broad, including those who work on building applied AI systems (i.e., SE practices), those who manage, productize, and otherwise "package" applied AI systems (i.e., business/management practices), and those who interact with AI systems as part of their everyday work (i.e., practices of use and interaction with enterprise AI systems). This ethnographic endeavor began in Fall 2017 and includes several sources of qualitative data, including semi-structured interviews (both formal and informal), participant observations, and artifact analysis.

This paper draws on analyses of four projects from this broader investigation: two long term (>12 months) projects; and two shorter term (<6 months) projects. The long-term projects focused on building a software/system service, while the short-term projects were exploratory in nature. Details on each are provided in Table 1. All proper names (including the names of informants and projects) are pseudonyms.

Data were analyzed inductively, following techniques similar to those in constructivist grounded theory [4]. Our analyses here are focused on identifying sensemaking practices. Qualitative coding revealed emergent, intermediary themes of ambiguity (*what is tha*t?), surprise (*that is different than what I expected*), and uncertainty (*I'm unsure*). While this work is preliminary, it is aimed with the eventual goal of developing a processual understanding of sensemaking in AI/ML SE work practices.

Project Pseudonym and description	**Time period**	**Sources of Data**
Alpha – software development project building an intelligent decision-support system to support the design work of IT architects, who design IT infrastructures	Dec. 2017 – Aug. 2019	Semi-structured interviews and usability testing (with members of the user community of IT architects) and participant observation of technical development work on this project; also design and execution of a user feedback program to facilitate exchange between user community and technical team (72 informants)
Beta – exploratory interview study of ML developers' experiences and work practices and design evaluation of an interface	June 2018 – Aug. 2018	Semi-structured interviews and participatory design of an interface for a novel, open-source AI/ML toolkit (13 informants)
Gamma – software development project building a system to support the model improvement practices of data scientists	Oct. 2018 – Sept. 2019	Semi-structured interviews and participant observations of model improvement practices; participatory design of a system to support those practices (8 informants)

Delta – exploratory interview study of those who work on natural language processing (NLP) projects	July 2019 – Sept. 2019	Semi-structured interviews (30 informants)

Table 1. Describes the four projects from the larger ethnographic study.

3 Findings: Sensemaking Practices

We organize our findings in three themes: making sense of AI/ML environments; making sense of AI/ML model ecosystems; and making sense of business contexts when building AI/ML systems.

3.1 Making Sense of AI/ML Environments

Working on AI/ML SE projects involves getting a handle on the overall development environment. This means understanding what algorithms and methods are being used in a project – for example, Neural Network, Support Vector Machine (SVM), or Decision Trees. In complex projects using experimental methods (typical in R&D teams), this can mean getting a handle on combo or hybrid approaches that weave together different algorithms. The Beta project, for example, involved a hybrid approach that took the activations from the last hidden, dense layer of a Neural Network and then clustered those activations using a k-means clustering algorithm. The AI method in the Beta project was part of an open-source AI toolkit to examine datasets for the presence of "poison" – intentionally tampered data. Based on the method's underlying research experiments, if one of the activation clusters was substantially smaller than the other, it was an indicator the small cluster contained tampered data. Even though developers understood the Beta method was a combination neural network/k-means algorithm, many questions still arose on the particulars of the clustering analysis. We can see such a concern as Frank, a developer, thinks aloud when reviewing the toolkit's interface:

> So I see cluster size is the indicator, but I'm wondering how are you guys computing these cluster sizes? Is it just like looking at the overall distribution of the clusters that you produce from the clusters? And also, what are the heuristics you are using to determine whether they're about the same size or whether they're big or small? (Frank, developer, Beta interview).

Questions like Frank's (about the particulars of the clustering approach used in the method) were raised by many developers in the Beta project. What this tells us is that in understanding an algorithm's mechanics, developers make sense of "an algorithm" at differing levels of granularity – while "k-means clustering algorithm" provides a general understanding of the method's algorithmic mechanics, in order to evaluate and derive meaning from any results it might display, developers need details of operations (and their significance) at a finer grain.

Another concern for ML developers in making sense of Beta's novel AI method was understanding the experiments and scenarios tested in a method's development process. Angie wondered aloud "*I wonder why they only use activations from the last layer, instead of the whole?*" (Angie, developer, Beta interview). Similarly, Laverne commented aloud as she interacted with the interface *"Hmm, interesting, okay, so this is empirical? These metrics are derived from experiments."* Here we see how understanding different decisions made in the process of developing the AI method helps developers assess its soundness, as Laverne continued, asking: *"And what if the cluster sizes were comparable? Like if you had that much poison in your dataset? If you had as much poison as clean, then it wouldn't even flag it, would it?"* (Laverne, developer, Beta interview). Understanding what experiments the method's inventors ran during its development was important to understanding its potential limitations, as Georgia, an ML developer, explained: "*So just putting, you know, stating 'We studied it in this context, using this classifier, and it's not as robust in this scenario versus that scenario' or maybe it is robust in both and that would be even more useful for me to know."* (Georgia, developer, Beta interview). Through these examples from the Beta project, we are able to see how developers must make sense of the algorithms being used (what algorithms are being used here?) but also how best to appropriately interpret the algorithmic method's output and its significance for the task at hand.

In addition to specific AI/ML algorithms and methods, developers must also make sense of the software libraries used in a given project. Examples of AI/ML software libraries include TensorFlow, Keras, PyTorch, or Apache Spark. Developers gain fluency in particular libraries, even garnering a favorite or preferred library. But even if an SE team is using the same library (e.g., Keras), the library can be variable across releases, causing issues with compatibility. There can be continuity issues with code written in one version of a library and then executed later using a subsequent version. Xavier raised this issue in an interview. He had recently joined the team and was trying to get his machine configured and onboarded to pick up his teammate's existing code. Xavier tried to execute the code on his machine but kept getting errors returned which puzzled both him and his teammate. "*We were both using Keras (a common, open source AI framework), so we were both like what is going on?,"* he explained. Perplexed, Xavier spent some time searching online forums but couldn't find anything particularly useful. "*Keras is known for not having the best documentation*," he explained, "*but usually I can find relevant discussions online*." Discouraged from the lack of insight online, Xavier felt stuck and shared his frustration with others who shared his workspace. "*That's when I was just talking about it in our room, asking if anybody else had this problem when they started*," Xavier said. He continued:

> It was kinda funny (laughs) after all those issues and trying everything I could think of, trying for several days, [Jesus, a

> colleague] was like 'Oh yeah, what version of Keras are you using? The most recent release caused a lot of portability issues.' And just like that, we figured it out. (Xavier, developer, Beta interview).

In these examples, we are able to see how developers must make sense of a project's broader AI/ML environment – simply knowing the project is dealing with Neural Networks or using the Keras library is only part of the equation. For SE work to proceed, developers must develop a deeper understanding of the broader AI/ML environment to effectively work in it.

3.2 Making Sense of AI/ML Model Ecosystems

The environments of applied AI systems are complex and dynamic – often, the overarching system/service can have many components and sub-processes. In addition to the AI/ML environment (e.g., algorithms and libraries, as discussed above), models are also a key site of sensemaking in AI/ML ecosystems. This can include pre-trained models, models that are trained elsewhere and then used by subsequent SE teams "off the shelf" or as a starting place to bootstrap off and build a bespoke, hybrid model. In a pre-trained model scenario, software teams must figure out what the pre-trained model entails, its scope and training data/subsequent limitations, as Andrej, a technical executive, said during an interview: "*When we talk about AI we talk about a lot of '-abilities,' interpretability, explainability. But another thing is the data, what data was this model trained on?"* Andrej went on to elaborate: *"That helps you interpret what it is giving you. It can be hard for people to interpret a lot of this if it's not clear about the data behind it."* (Andrej, technical executive, Delta interview). This was echoed in many developers' accounts as well, as they described working with pre-trained models. Vincent, a developer recounted: "*We really had to dig around to understand what the pre-trained model could do, before we could come up with some novel research to build on top*" (Vincent, developer, Delta interview). He then described the testing procedures the SE team devised to create benchmarks to measure any subsequent tweaks they made to the off-the-shelf model. Any changes or improvements to the model or overall ML architecture had to be carefully evaluated for impacts to speed/performance of the resulting user experience (UX) of the service: "*Even adding seconds, you know, even half seconds to run time can cause problems*," Vincent's teammate and fellow developer, Jiro, stated, "*so we are always testing those impacts before we implement any changes*." (Jiro, developer, Delta interview). Thus, SE teams must make sense of pre-trained models (what data are they trained on, what can they do and not do), as well as any subsequent changes or modifications they make to the model and its significance for the overall performance of the service the AI will be embedded in.

In addition to pre-trained models, some projects might use other off-the-shelf knowledge objects, such as word embeddings or knowledge graphs. These types of objects are different from pre-trained models, yet also require developers to carefully understand their composition and limitations as Kurt shared: "*Word embeddings are quite biased*" (Kurt, developer, Delta interview). He went on to list some common examples to illustrate his point, associating "attorney" with male, yet "paralegal" with female. SE teams need to identify these types of bias, which can be difficult when using models and knowledge objects created elsewhere. "Created elsewhere," though, can also mean models and code written by other SE teams or written by earlier iterations of a software team, who "inherit" ecosystems and must make sense of what has already been done. "*We didn't start this project from scratch*," Xiao said, "*there was a legacy from previous teams. Then we have some discussions on how we wanted to change some things and keep some things*." (Xiao, developer, Delta interview). Software teams must iteratively make sense of ecosystems which are dynamic and include legacy components with varying temporalities [6], involving artifacts from a variety of sources both external and internal to the team and organization.

SE teams must also make sense of multi-model AI/ML ecosystems. Many applied AI systems will include more than one model in the overarching digital service, for example a text classification task also requires processing text inputs through parsing, which can be done via an "upstream" extraction/parsing model that feeds into the classification model. Alternate configurations might architect these two tasks to be housed within a single model, yet there will likely be different developers working on each task. This requires establishing internal team infrastructure to smoothly coordinate and collaborate on future model refinements – developers must understand the dependencies between the different tasks and/or different models. The Gamma project was focused on building a system to support error analysis – the process of analyzing individual model errors to identify error root causes and plan subsequent model improvement work. Typical model errors include precision and recall errors, but there may also be other errors, such as mislabeled ground truth or a preceding model's error (e.g., a parser error, rather than a classification error). In enterprise settings, where AI/ML software services enhance complex domain activities like legal contract analysis, feedback from users (typically subject-matter experts or "SMEs") is integral in identifying model errors. Early in the life of a model, SE teams will work closely with a handful of SMEs in building and refining the service. Yet later, after the service is launched, feedback can come in from a growing and heterogeneous group of end-users. Effectively and efficiently processing user feedback is essential in the maintenance and refreshment of the service, yet the dynamic and complex nature of the AI/ML ecosystem requires ongoing sensemaking and infrastructuring work by SE teams.

3.3 Making Sense of Business Contexts When Building AI/ML Systems

Sensemaking and infrastructuring work in AI/ML ecosystems extends beyond technical issues and also includes concerns that emerge between the business context and the AI/ML service as SE work unfolds. These concerns are consequential both for the ongoing technical work of coding and building, as well as broader project/stakeholder management. "*The key to being successful [in building ML] is all about thinking through the problems or questions you want to work on*," Kai explained, "*one that fits appropriately with your data. Once you have that clear in your mind, then you have an easier time making choices about different ML techniques*." He reflected on the recent explosion of interest in ML: "*there's a lot interest in ML today, but I always stress that yes, it's a hot topic, but it's like any other technical sub-field*," he explained, "*it's got to make sense for what you are trying to do*." (Kai, developer, Beta interview). Having clarity within the SE team on the appropriateness of a problem/task match is important because the team will need to help clients think through the service's match with their organizational objectives. *"You want to tell clients there is no magic in the model,"* Svetlana recounted. *"We want to ask them: 'What are you looking to do in your organization? What are you trying to do? What's your data and what's your task?'"* She continued, reflecting: *"We don't want to try a pre-trained model off-the-shelf if it's not a good fit, because then they will be disappointed. It's got to be aligned with the business problem."* (Svetlana, developer, Delta interview).

Having these types of conversations with clients requires making sense of "how technical" various stakeholders are on a project and adjusting the software team's communication approach accordingly (e.g., use of technical jargon, etc). "*From my experience*," Emil shared, "*99% of the time, even just saying we used a 'deep learning neural network' is too much detail for conversations with product*." (Emil, developer, Delta interview). Natasha, an offering manager on an AI/ML service, shared a similar perspective: "*Real life clients are not data scientists, they are business people and technology leaders who need to implement some AI inside of some traditional software platform. And they want an AI service to bolster their product without having to invest in AI*." She elaborated: "*If a data scientist was brought into the room, … if a data scientist was involved in the purchase decision, more technical kinds of questions would come up. But the questions we tend to see are more business-oriented, how do we integrate this with more deterministic software code, how do we integrate it into our business*." (Natasha, service offering manager, Delta interview).

SE teams must develop a fluency with the client's industrial domain and business processes to appropriately navigate it. This is an ongoing process of sensemaking and improvisational learning, adjusting day-to-day practice as lessons are learned. For example, Jane talked of her journey to working with SMEs and leveraging their industrial knowledge to guide model development: "*At first, I'd just dump all my features into a model, but then you don't get anything good*," she stated. "*So now I work with an SME on what are the top 3 or 4 features you'd look for first, then use that to guide the feature engineering to make sure we are looking at the right things."* She reflected on the benefits of this close, collaborative approach: *"Then that becomes naturally more interpretable to the SME because it's things they would look for, and know*" (Jane, developer, Delta interview). Making sense of business/AI coupling can also involve thinking through if and how the AI/ML service will alter the business process it will become embedded in. For example, Chandrasekhar was working on a project building an AI/ML service in the pharmaceutical domain. The service worked to identify "key opinion leaders," a typical activity in marketing. Chandrasekhar talked of the software team's work in explaining how the AI/ML approach to identifying key opinion leaders differed from the business-as-usual process (a manual, hand-curated process). "*We have to really work with clients to understand the differences,"* he said, "*the AI service is much faster than the manual process, which is a key benefit, but it also has limitations in that there isn't a human verifying everything along the way*." (Chandrasekhar, developer, Delta interview). In other cases, clients may present the software team with constraints that the overarching business process remain unchanged, which can create unique SE considerations. For example, Sandeep was working on a project building an AI/ML service in facilities management. "*They were very clear, they did not want to have to create new policies or processes to integrate the service into their business*," he recounted. "*It really made us have to think, we had to understand their current process really well to ensure whatever we built would be amenable to it*." (Sandeep, developer, Delta interview). Many services, though, are intentionally aimed to catalyze organizational change – yet despite this intentionality, exactly what will be changed and how can remain ambiguous (or morph) during a project.

Even more challenging is understanding the broader repercussions of such changes – and defining what elements of a process must remain the same to ensure overall system (as well as organizational) stability. For example, the Alpha project was building an AI/ML decision-support system to augment the design work of IT architects who design IT system architectures. Their design work involves culling IT requirements from Request for Proposal (RFP) documentation and then matching those requirements to sets of offerings in the TechCorp catalogue. When implemented into an AI/ML system, these activities were accomplished through an NLP model (handling text extraction and classification) and then a downstream optimization model (matching the classification results to offerings and recommending an optimal solution). Explaining this complexity to users was a difficult task, made even more challenging as the system's architecture and underlying AI/ML functionality evolved over time in the project's Agile trajectory. As new requirements emerged from the user community, changes were made – yet this evolution raised new questions on reliability. For example, as the project progressed, the decision was made to integrate the system with

other TechCorp databases that IT architects used regularly in their design work – if the Alpha system was meant to be the "one stop shop" for their design work, it needed to also integrate a TechCorp database called OrgFrame. The connection between OrgFrame and the Alpha system's internal AI/ML models was accomplished via an API. Yet when this functionality was launched in the Alpha system, users raised a number of questions and concerns over the frequency and verifiability of the API's data stream. *How do I know it is drawing on the most current version of OrgFrame?* Investigating this question, the SE team uncovered a mire around the database, including its *ad hoc* refreshment approach which made it difficult to anticipate and appropriately architect to provide data inputs into the Alpha project's AI/ML models. In this example, we see how software teams must not only make sense of a project's broader industrial and business domain as a project starts, but that these are evolving and dynamic sites of sensemaking, which are consequential in shaping if and how the team's work proceeds.

4 Discussion and Conclusions

In this paper, we have investigated the topic of sensemaking in applied AI/ML software engineering projects. As our empirical findings illustrate, AI/ML software projects are dynamic and complex settings which necessitate software teams' active and engaged sensemaking as they strive to create coherence of: AI/ML environments (e.g., algorithms/methods and libraries), AI/ML model ecosystems (e.g., pre-trained models and "upstream" models), and the business contexts that emerge while building AI/ML systems (e.g., how the AI/ML service relates to the business problem at hand). Our paper builds on recent scholarship noting the integral role of sensemaking in SE work [7, 16-18]. Our focus on AI/ML projects also contributes empirical insights to the nascent body of research investigating the challenges of software work involving AI/ML capabilities [2, 10, 12]. Ehlers [7] characterizes the work of software development as "managed sensemaking," noting a dynamic interplay that unfolds in everyday SE work practices that is at once tactical, yet also improvisational. It is tactical in that it is expected and often planned for – onboarding new members to the software team (*What libraries are we using? What legacies are we inheriting?),* starting new projects with clients (*What are they looking to do with AI? What is the business context?*), and so on. But it is also improvisational and situated – as we have seen, the need to make sense of ambiguous, uncertain, and surprising events is ongoingly emergent as software projects proceed – *What version of the library are you using? What are the limitations of using open source objects in our system? How might new requirements challenge our existing architecture?*

As we wrap up, we note two implications of our work that prompt further inquiry – one is *emergence* and another is *resonance*. By emergence [13], we mean the fluid nature of sensemaking practices and by resonance, we mean the relationship between our findings in AI/ML projects and the sensemaking practices of software teams in traditional projects. We have organized our empirical findings into three themes; these divisions are analytical and in practice, their boundaries may overlap, blur, merge, or cleave in different configurations of entities [15]. This presents both a challenge and an opportunity for those interested in developing SE process models of sensemaking – how and in what ways do software teams themselves understand their sensemaking practices? Do the analytical contours of "AI/ML environment," "AI/ML model ecosystem," and "business context" feel true-to-life and useful for them? Who else must make sense of AI/ML ecosystems? How and when do they contribute to the work of software teams? How might we capture these fluid and organic dynamics in process models of sensemaking? Further, we note that many sites of sensemaking we have explored here are not wholly "new" and instead echo existing insights from research into the collaborative, human aspects of SE. This presents an opportunity for synthesis and coalition – What is new about AI/ML software engineering? What is familiar and what persists from existing SE scholarship? Such questions are ripe for future scrutiny.

ACKNOWLEDGMENTS

Thank you to informants for sharing your time, experiences, and insights. All opinions expressed herein are our own and do not reflect any institutional endorsement. Dr. Paine's work is supported by the U.S. Department of Energy, Office of Science and Office of Advanced Scientific Computing Research (ASCR) under Contract No. DE-AC02-05CH11231.

REFERENCES

[1] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi and T. Zimmermann. 2019. Software Engineering for Machine Learning: A Case Study. in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 291-300. 10.1109/ICSE-SEIP.2019.00042

[2] A. Begel. 2019. Best Practices for Engineering AI-Infused Applications: Lessons Learned from Microsoft Teams. in *2019 IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI) and 6th International Workshop on Software Engineering Research and Industrial Practice (SER&IP)*, 1-1. 10.1109/CESSER-IP.2019.00008

[3] Matthias Book and André van der Hoek. 2018. Sketching with a purpose: moving from supporting modeling to supporting software engineering activities. in *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering*, Gothenburg, Sweden, Association for Computing Machinery, 93–96. 10.1145/3195836.3195854

[4] Kathy Charmaz. 2014. *Constructing Grounded Theory: A Practical Guide Through Qualitative Analysis*. Sage.

[5] Souti Chattopadhyay, Nicholas Nelson, Thien Nam, McKenzie Calvert and Anita Sarma. 2018. Context in programming: an investigation of how programmers create context. in *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering*,

Gothenburg, Sweden, Association for Computing Machinery, 33–36. 10.1145/3195836.3195861

[6] Marisa Leavitt Cohn. 2016. Convivial Decay: Entangled Lifetimes in a Geriatric Infrastructure. in *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, San Francisco, California, USA, ACM, 1511-1523. 10.1145/2818048.2820077

[7] Kobus Ehlers. 2011. Agile software development as managed sensemaking, Stellenbosch: University of Stellenbosch.

[8] Rob Fuller. 2019. Functional organization of software groups considered harmful. in *Proceedings of the International Conference on Software and System Processes*, Montreal, Quebec, Canada, IEEE Press, 120–124. 10.1109/icssp.2019.00024

[9] Valentina Grigoreanu, Margaret Burnett, Susan Wiedenbeck, Jill Cao, Kyle Rector and Irwin Kwan. 2012. End-user debugging strategies: A sensemaking perspective. 19 (1). Article 5. 10.1145/2147783.2147788

[10] F. Ishikawa and N. Yoshioka. 2019. How Do Engineers Perceive Difficulties in Engineering of Machine-Learning Systems? - Questionnaire Survey. in *2019 IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI) and 6th International Workshop on Software Engineering Research and Industrial Practice (SER&IP)*, 2-9. 10.1109/CESSER-IP.2019.00009

[11] Sami Jantunen, Rex Dumdum and Donald C. Gause. 2019. Towards new requirements engineering competencies. in *Proceedings of the 12th International Workshop on Cooperative and Human Aspects of Software Engineering*, Montreal, Quebec, Canada, IEEE Press, 131–134. 10.1109/chase.2019.00038

[12] F. Khomh, B. Adams, J. Cheng, M. Fokaefs and G. Antoniol. 2018. Software Engineering for Machine-Learning Applications: The Road Ahead. *IEEE Software*, 35 (5). 81-84. 10.1109/MS.2018.3571224

[13] Charlotte P. Lee and Drew Paine. 2015. From The Matrix to a Model of Coordinated Action (MoCA): A Conceptual Framework of and for CSCW. in *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, Vancouver, BC, Canada, ACM, 179-194. 10.1145/2675133.2675161

[14] Lucy Ellen Lwakatare, Aiswarya Raj, Jan Bosch, Helena Holmström Olsson and Ivica Crnkovic. 2019. A Taxonomy of Software Engineering Challenges for Machine Learning Systems: An Empirical Investigation. in, Cham, Springer International Publishing, 227-243.

[15] Drew Paine and Charlotte P. Lee. 2020. Coordinative Entities: Forms of Organizing in Data Intensive Science. *Computer Supported Cooperative Work (CSCW)*. 10.1007/s10606-020-09372-2

[16] Paul Ralph. 2015. Developing and evaluating software engineering process theories. in *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, Florence, Italy, IEEE Press, 20–31.

[17] Paul Ralph. 2015. The Sensemaking-Coevolution-Implementation Theory of software design. *Science of Computer Programming*, 101. 21-41. https://doi.org/10.1016/j.scico.2014.11.007

[18] Paul Ralph and Rahul Mohanani. 2015. Is requirements engineering inherently counterproductive? in *Proceedings of the Fifth International Workshop on Twin Peaks of Requirements and Architecture*, Florence, Italy, IEEE Press, 20–23.

[19] Walt Scacchi. 2002. Process Models in Software Engineering. in Marciniak, J.J. ed. *Encyclopedia of Software Engineering*.

[20] Todd Sedano, Paul Ralph and Cécile Péraire. 2019. The product backlog. in *Proceedings of the 41st International Conference on Software Engineering*, Montreal, Quebec, Canada, IEEE Press, 200–211. 10.1109/icse.2019.00036

[21] H. Sharp, Y. Dittrich and C. R. B. de Souza. 2016. The Role of Ethnographic Studies in Empirical Software Engineering. *IEEE Transactions on Software Engineering*, 42 (8). 786-804. 10.1109/TSE.2016.2519887

[22] Ben Shreeve, Paul Ralph, Pete Sawyer and Patrick Stacey. 2015. Geographically distributed sensemaking: developing understanding in forum-based software development teams. in *Proceedings of the Eighth International Workshop on Cooperative and Human Aspects of Software Engineering*, Florence, Italy, IEEE Press, 36–42.

[23] H. Washizaki, H. Uchida, F. Khomh and Y. Guéhéneuc. 2019. Studying Software Engineering Patterns for Designing Machine Learning Systems. in *2019 10th International Workshop on Empirical Software Engineering in Practice (IWESEP)*, 49-495. 10.1109/IWESEP49350.2019.00017

[24] Karl E. Weick. 2009. *Sensemaking in organizations*. Sage, Thousand Oaks, CA.

[25] Christine T. Wolf. 2020. AI Models and Their Worlds: Investigating Data-Driven, AI/ML Ecosystems Through a Work Practices Lens. in, Cham, Springer International Publishing, 651-664.

[26] Christine T. Wolf and Jeanette L. Blomberg. 2020. Making Sense of Enterprise Apps in Everyday Work Practices. *Computer Supported Cooperative Work (CSCW)*, 29 (1-2). 1-27. 10.1007/s10606-019-09363-y