Learning Observability Tracing Through Experiential Learning

Anders Sundelin[1][0000−0001−9898−2222]

Blekinge Institute of Technology, Karlskrona, Sweden anders.sundelin@bth.se

Abstract. In a large-scale software development product development organization, we found that most developers, although experienced, were lacking architectural knowledge of the specific developed product.
As a remedy, we evaluated whether we could stimulate learning the product architecture by conducting training in how to use the product's distributed tracing platform, built on the OpenTelemetry standard and the open-source Jaeger Tracing visualization tool.
We planned and participated in a training event, where parts of the organization explored, using experiential learning, how to set up and use tracing to troubleshoot a realistic fault scenario we prepared. Respondents were asked to rate the tool according to the Technology Adoption Model (TAM), and responses were collected on Likert-type scales, analyzed, and summarized using a Bayesian workflow.
Even as tool usage post-training was low, respondents still had a positive attitude toward using the tool, valued the experiential training, and expressed a strong intent to use the tool for program comprehension.

Keywords: Micro-services · Observability · Distributed tracing · Experiential learning.

1 Introduction

Even with comprehensive design documentation, becoming fluent in a codebase with millions of lines is difficult. Feature location involves static (source-code-based) and dynamic (logging, tracing) methods [3], but trace-based call-flow visualizations require setup and usage skills. With the advent of distributed microservices, where one request passes through multiple services before returning, the problem of locating features is even more complex, and distributed tracing tools have emerged to facilitate program comprehension and fault localization [2].

Over six months, we conducted a three-cycle action research study in a large-scale software development organization, encompassing around 100 developers (15 teams) on multiple sites in two time zones (Europe and India). As the majority of developers were relatively new to the large code base (more than 12 MLOC Java code), dating back 15 years, we wanted to understand if a distributed tracing tool could be used to help developers understand the program flow. To test this theory, we planned and executed a learning session in which 19 developers, on two sites, learned how to set up and use the Jaeger tracing tool in a realistic problem scenario. Throughout the study, we collected feedback via surveys from the whole organization on how they perceived their competence and their experience with the training.

This paper is structured as follows: This section contains the overall problem formulation and the research questions. Section 2 contains background and related work, while Section 3 includes the research design, studied context, and methodology. Section 4 presents the results of our action cycles, which are condensed into learnings in Section 5, before we end the paper in Section 6 with conclusions.

1.1 Problem Statement and Research Questions

In our studied case, even as the product had supported tracing for over six months, few developers used it to learn about the product, which leads us to our problem formulation:

Can we use a tracing tool, like Jaeger, to increase architectural knowledge of a large product in a development organization where most developers lack specific product experience?

To investigate this problem, we formulated the following research questions:

(RQ1): To what extent do developers perceive that tracing tools help them understand the system architecture of a large-scale, multi-service system, most of which has been written by other developers?

(RQ2): How do developers perceive learning Jaeger tracing via experiential learning, using the system where they usually work?

(RQ3): To what extent do developers intend to continue using the tracing tools?

2 Background and Related Work

The Open-Source standard and framework OpenTelemetry[1] was formed to enable practical observability of microservices in an easy, universal, vendor-neutral, loosely coupled, and built-in way. To meet these goals, it provides standards and reference implementations that generate, collect, and export telemetry data in traces, metrics, and logs. Jaeger Tracing[2] is an open-source tool that produces web visualizations of collected OpenTelemetry data.

Gortney et al. [5] conducted a systematic mapping study and found that dynamic tracing, log analysis, and metrics collection were the three main practices used to visualize microservice architectures.

To assess the state of observability tools in industry, Li et al. [7] conducted an interview study, where 25 interviewees from ten companies of different sizes and in five domains were interviewed about their tracing practices and tools. All but the smallest of the studied companies (≈ 10 services and ≈ 20 kLOC) were employing, or considering using, some trace and log processing pipeline. The trace function was most commonly used for timeline and root cause analyses.

[1] https://opentelemetry.io/

[2] https://www.jaegertracing.io/

3 Research Design

3.1 Context

The organization consists of 15 cross-functional teams of $6-8$ developers, equally split between Europe and India. The product runs on Kubernetes in the cloud, and includes both in-house and externally sourced services, mostly Java (3.01 MLOC production, 2.22 MLOC unit tests, 6.49 MLOC functional tests), plus smaller Vue.JS (181 kLOC) and TypeScript (817 kLOC) front ends.

Half of the 15-year-old code base is older than five years, and only 30% is written by current employees, with the most experienced team contributing 8.8%, and average team contributions under 1%. A survey (55% response rate) shows both European and Indian developers average over ten years in general software development experience, but those in Europe have more product knowledge (average 4.2, max 15 years) than those in India (average 2.5, max 4 years).

3.2 Summary of Research Cycles

The organization asked us to investigate why OpenTelemetry and Jaeger were not used, even though architects saw them as valuable for exploring product architecture and unfamiliar features.

Cycle 1: Problem identification and formulation assessed the organization's development practices by collecting system statistics and administering a survey[3], asking respondents to rate their skills in six development areas, using a $0-100$ scale with descriptive anchors for five levels.

Cycle 2: Planning and executing the training session involved developing and delivering Jaeger training simultaneously at the European and Indian development centers. The session was one of six optional tracks during an organization-wide training day, with instructor-led, in-office participation to promote skill sharing and cross-team networking, while organizers allocated attendees based on budget and preferences.

Cycle 3: Follow-up of the training examined whether developers used Jaeger after the training by monitoring internal wiki page views. After one month, a follow-up survey was distributed to gather feedback on the training experience.

3.3 Data Collection and Analysis Methods

All our surveys used 7-level symmetric Likert scales with mandatory items and optional qualitative questions to assess agreement with survey statements. We applied Bayesian data analysis [4, 8] under the TAM framework to model constructs like usability, ease-of-use, and intent-to-use, comparing our treatment group to two control groups from unrelated training sessions. Models were ranked using PSIS-LOO [11] and built with the brms framework [1], interfacing with the Stan language, with full methodology available in the replication package [10].

[3] Available in the replication package: `https://doi.org/10.5281/zenodo.15678405`

We compared training sessions and sites by analyzing distributions, means, and credible intervals of the expected rating (defined as $\mathbb{E} \equiv \sum_{k=1}^{7} p(k) \times k$, where $p(k)$ is the probability of rating k) for the different constructs. We also summarized the qualitative feedback and presented this to the organization.

4 Results and interpretation

We follow recommendations from Staron [9] and report the results per cycle.

Cycle 1: Problem identification and formulation focused on assessing the current situation, from a tracing and observability perspective. We administered a survey to assess the self-rated competence level in the organization from various development perspectives. While the majority of the 55 respondents, in both Europe and India, are confident in writing and debugging Java code, fewer were confident in Vue.JS, and only four individuals (all based in Europe) reported that they were confident in Jaeger tracing.

We presented these findings, together with the proposed training, to the organization, which agreed to continue the study.

Cycle 2: Planning and executing the training session started with collecting background knowledge and planning an interactive training session. We wanted to involve participants and use a realistic scenario recognizable to developers. The cycle lasted six weeks, with regular follow-up meetings, comprising four phases: (i) planning and scoping the tutorial, (ii) preparing the setup instructions and developing the scenarios the students would face, (iii) validating the instructions (also including fixing issues found in existing documentation), and finally (iv) executing the training sessions, and collecting direct feedback.

Our planning phase focused on how to introduce a condition that would trigger a suitable anomaly (e.g., prolonged response time). We developed one mandatory "seeded" problem and four "overflow" tasks, freely chosen by each team. To aid teams, we developed four hints for our problem, where the first hint indicated the product area, and the last one pointed to a complete functional test case reproducing the problem. Depending on the team's problem-solving progress, teachers distributed hints as text on printed paper. All teams needed the first hint, and one team used all four.

After one experienced developer formulated and implemented the seeded problem, including hints, we used two validation phases. First, one developer on each site independently followed and commented on the instructions. After adjustments and clarifications of instructions and hints, one junior developer performed final validation, including measuring the required time.

The training day comprised five one-day training sessions, from requirements engineering to security analysis. Participants ranked their preferences, and organizers allocated spots based on these choices and available capacity. All participants in the *Generative AI* and Vue.JS tracks had chosen these as their primary alternative. All architect and developer teams were represented in the *Jaeger tracing* track, but only 70% had ranked it as their primary choice.

After the training day, we collected opinions on the training from participants on both sites via a prepared survey. The tool-centric training sessions shared the same TAM-based structure of the questions, but had different characteristics:

Generative AI Participants in this track would learn principles behind Retrieval-Augmented Generation (RAG), use and partially develop a Python-based RAG solution in an existing Jupyter Notebook. Developers currently use none of these tools in their daily work.

Jaeger tracing This is our treatment group, where we explain how to set up OpenTelemetry and Jaeger in an experiential learning setting, and use these tools to solve a realistic, prepared problem in a special product branch.

Vue.JS Participants in this track would learn how to leverage the (for the product) new Vue.JS component pattern[4]. They were expected to learn the pattern and refactor existing views by developing and using common components in their normal development environment.

We asked the participants to rate the tools they used in their training from the angles of: (i) usability, (ii) ease-of-use, (iii) ease-of-access, and (iv) intent to use the tool in the future. Overall, the ratings were positive and similar for both sites, indicating that the training was well received.

Table 1. Respondents per training session, their reported years of professional experience, and the percentage of participants responding to the survey.

			Years of prof. experience					**Response**
	Site	**N**	**Mean**	σ	**Median**	**Min**	**Max**	**Rate**
GenAI	Europe	3	14.0	14.2	9	3	30	50%
	India	6	13.7	3.6	13	10	20	75%
Jaeger	Europe	10	17.5	8.7	17.5	8	30	83%
	India	9	12.9	3.4	14	5	17	100%
Vue.JS	Europe	5	14.8	10.2	19	0	25	83%
	India	5	7.8	4.0	10	3	12	71%

Table 1 shows the number of respondents per session and site, and summary statistics on their (self-reported) professional developer experience. All sessions involved experienced professional developers, with a median experience ranging between 9 and 19 years. The response rate varied between 50% and 100%.

Figure 1a) shows the expected average usability rating, distribution, point estimate, and 95% credible interval. The figure shows that Vue.JS participants perceived the strongest usability, especially the Indian cohort. However, the European participants are *Quite Likely* to agree that Vue.JS components are usable.

[4] https://vueschool.io/articles/vuejs-tutorials/5-component-design-patterns-to-boost-your-vue-js-applications/

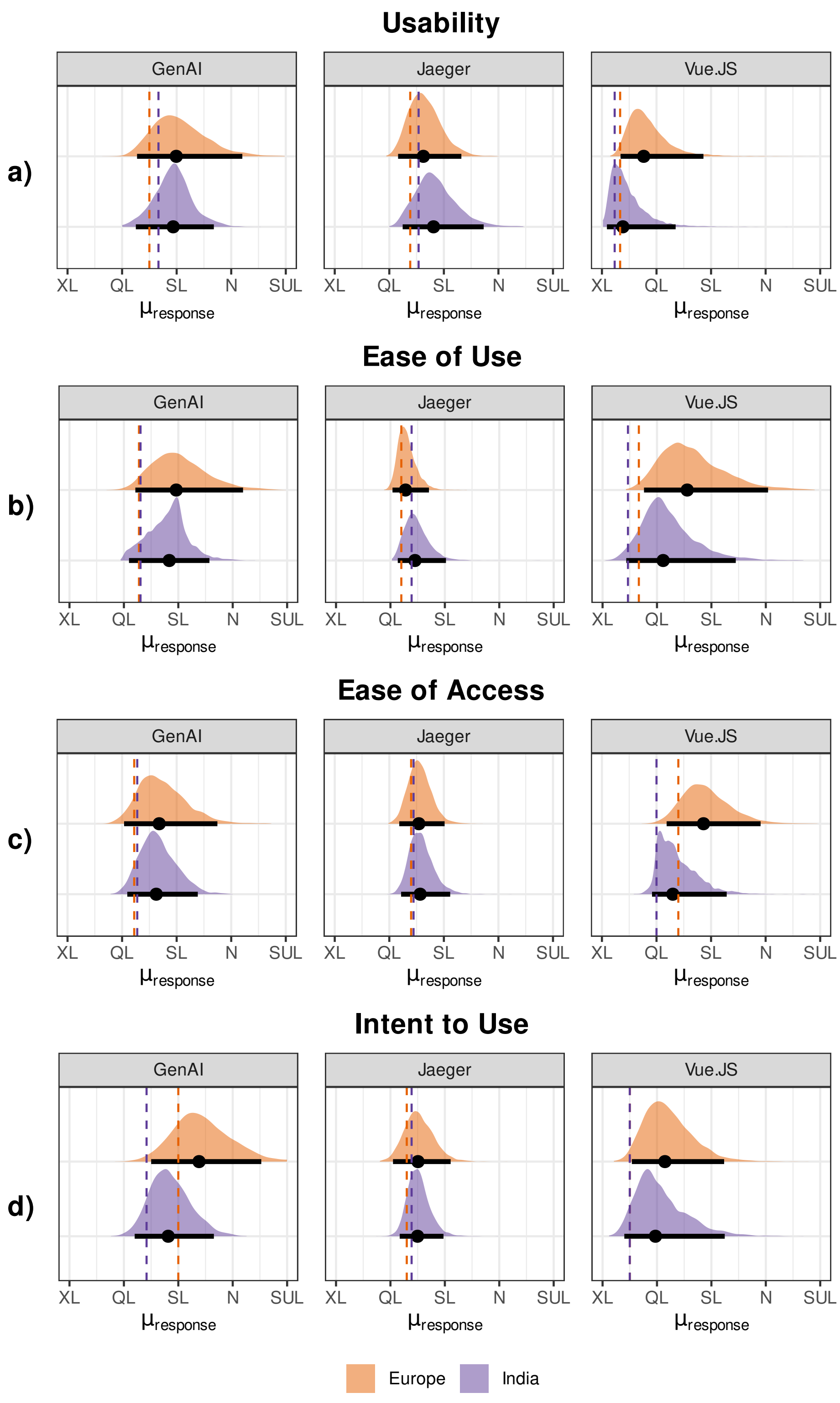

Fig. 1. The distributions, by tool and training site, of the posterior expected rating for the four constructs: usability, ease-of-use, ease-of-access, and intent-to-use. Likert levels on the x-axis represent *Extremely Likely*, *Quite Likely*, *Slightly Likely*, *Neutral*, and *Slightly Unlikely* agreement, omitting the last two levels. The point estimate of the expected rating is shown as a dot, surrounded by the 95% credible interval. Sample (response) means are shown as dashed lines.

Both cohorts of Generative AI participants agree that the RAG tool is *Slightly Likely* to be usable, though the European cohort also tends towards *Neutral.* The Jaeger participants take the middle ground, rating the tool between *Quite Likely* and *Slightly Likely* to agree on its usability. The Indian cohort leans more towards *Slightly Likely*, though the difference is negligible.

Concerning ease-of-use, Figure 1b) show that the Vue.JS participants are more skeptical, particularly the European cohort, which rates ease-of-use *Slightly Likely*, while the Indian cohort leans more towards *Quite Likely.* Both the other tracks are very similar between sites. However, the Jaeger distribution is slightly narrower, and estimates, with 95% probability, that the expected value for ease-of-use lies between *Slightly Likely* and *Quite Likely.* Although the point estimate of the Generative AI track is similar, the distribution is wider, meaning there is more uncertainty about where the expected value lies, particularly towards the *Neutral* level.

Figure 1c) shows the perception of how easy the tools are accessed in day-to-day use. All tools rate between *Quite Likely* and *Slightly Likely*, though the European Vue.JS participants lean closer towards *Slightly Likely*, while their Indian counterparts lean more towards *Quite Likely.*

Finally, as shown in Figure 1d), we measured the intent to use the tool in the future. Not surprisingly, the Vue.JS participants were most certain that they would use Vue components, with the expected value close to *Quite Likely* for both European and Indian cohorts. The Generative AI participants were more skeptical, particularly the European cohort, where the expected rating was estimated to lie between *Slightly Likely* and *Neutral.* The Indian cohort was more positive, though close to *Slightly Likely.* Both Jaeger groups were equally certain that they lean between *Slightly Likely* and *Quite Likely* to continue using the tracing tool in the future.

Fifteen of the 19 participants in the Jaeger training also provided qualitative responses. The thirteen positive responses range from: "Knowing that the tool exists" to the more explicit: "Save significant debugging time when you have visibility on complete code flow across components along with their respective process time." Five respondents found things to improve related to the training (e.g., "Docker logins should be fully working"), and six respondents commented on the Jaeger GUI (e.g., "I found the comparison feature difficult to understand").

We also surveyed the teachers, achieving between 50% and 100% response rates for the tool sessions. The most positive teachers were found in the Jaeger track, where teachers indicated "it went generally well," and "pretty much smooth." The primary Vue.JS track teacher indicated that the session probably had "too much freedom," and that "letting people choose what they want to do can be overwhelming." The teacher indicated that more structure and options would be imposed if the track were held again. The single (out of two) AI teacher respondent indicated that developers requested "more post-practices," indicating that the participants wanted more experience with the tool they partly developed. The Jaeger tracing and Generative AI teachers were *Extremely* or *Quite* agreeing that they had enough time and resources for preparations and that the tasks were adequate for the participants. In contrast, one Vue.JS teacher was only *Slightly* agreeing to those statements. However, all teachers enjoyed being mentors and agreed that participants learned something useful.

Cycle 3: Follow-up of the training investigated Jaeger tool usage after the training by analyzing page views of the setup instructions on the corporate wiki[5]. Due to low page traffic, we conducted a follow-up survey one month later to assess how participants applied their learnings and identify barriers to tool adoption.

The survey had a low response rate, with only 23 replies, including six Jaeger training participants, none of whom reported using the tool. However, they expressed motivation to use tracing for learning program flow, and slightly agree that it would improve their understanding and development speed. All six respondents provide qualitative feedback (e.g., "I will use it when needing to trace something that is hard to track down in some other way").

Conclusion Related to **RQ1**, respondents have a favorable view of the utility of a tracing toolchain like OpenTelemetry and Jaeger to aid program comprehension. The expected intent to use the tools is close to "*Quite Likely*" and our model is 95% sure that the intent to use is higher than "*Slightly Likely*". This is also confirmed by the qualitative data collected via the survey.

Related to **RQ2**, we find from qualitative survey responses that the respondents view the training session positively, though two respondents indicate they had to overcome troubles with the used environment. No respondents stated that the problem was too complex or that the hints gave away too much. The participants appear engaged, and they seem to appreciate the preparations.

Related to **RQ3**, web access logs indicate that the setup instructions were rarely accessed, and survey responses, although limited, state that no respondent had used the Jaeger tracing tool one month after training. However, all six respondents reported that they were positive and would use it when they needed to learn more about the product architecture. This might indicate that the tool's utility is real, but the need to use it is infrequent. We intend to follow up on how the tool is perceived and how the competence spreads in the organization.

5 Learnings

5.1 Contribution to Theory

Our study indicates that tracing tools, such as OpenTelemetry and Jaeger are valuable for visualizing system architecture and localizing features. However, despite developers recognizing their usefulness, usage remains limited six months after introduction and one month after training, suggesting these tools are used sparingly, mainly when exploring new domains or unfamiliar features.

[5] `https://www.atlassian.com/software/confluence`

The experiential learning event, featuring one structured track and four "open-ended" problems, was well received, with all respondents providing positive feedback on their learning. Teachers also noted that they enjoyed guiding participants and making new social connections.

5.2 Recommendations to Other Companies

We structure our recommendations according to the study phases:

Prepare problem(s) well in advance, and validate (using "fresh" developers) that the problem is attainable, but still challenging. If possible, frame tasks in the regular code base, not a synthetic toy problem.

Problems can be "open" or "closed", and participants appreciated the "closed path" in the beginning, while enjoying the "open tasks" when they had learned more. To gain confidence in problem complexity, we used a representative developer to solve the "closed" problem, reviewing the number of hints needed and their detail.

Hints should be at different levels of detail, and we used paper slips rather than digital web pages. Teachers need to be prepared to disseminate these among the training groups, according to where they are in the learning journey.

6 Conclusions and Future Work

Our study adds to the findings by Li et al. [7] and Gortney et al. [5] that tracing can be an important tool to discover and learn about the product architecture. However, even as training participants perceived the tool and training positively, we found that these tools appear to be used relatively infrequently, possibly because developers rarely venture into unknown components.

We created a "Goldilocks problem," that was challenging yet attainable to help students focus on learning the tool chain, and used three developers to validate that the instructions were clear and unambiguous. Both students and teachers appreciated the initial structured approach, along with the four open-ended problems that allowed students to explore the tool freely.

Experiential learning [6] relies on attentive, self-directed students, so we provided hints of varying explicitness to participants as needed during the training. Survey responses show students found training tied to their daily work more usable, with higher scores for the Vue.JS component and tracing tool tracks than for Generative AI.

Despite developers expressing positive intent, one month after the training, none of our respondents had used the tracing tool in their daily work. This suggests that longer studies are needed to properly assess its adoption. We plan to continue surveying developers over time to monitor their work practices.

Acknowledgments. This research was supported by the Knowledge Foundation through the KKS Profile project SERT 2018/010 at Blekinge Institute of Technology, Sweden.

Disclosure of Interests. The author declares that he has no competing interests in the methods or tools mentioned in this article.

Bibliography

[1] Bürkner, P.C.: brms: An R package for Bayesian multilevel models using Stan. Journal of Statistical Software **80**(1), 1–28 (2017). `https://doi.org/10.18637/jss.v080.i01`

[2] Cassé, C., Berthou, P., Owezarski, P., Josset, S.: Using distributed tracing to identify inefficient resources composition in cloud applications. In: 2021 IEEE 10th International Conference on Cloud Networking (CloudNet). pp. 40–47 (2021). `https://doi.org/10.1109/CloudNet53349.2021.9657140`

[3] Dit, B., Revelle, M., Gethers, M., Poshyvanyk, D.: Feature location in source code: a taxonomy and survey. Journal of Software: Evolution and Process **25**(1), 53–95 (2013)

[4] Gelman, A., Hill, J.: Data Analysis Using Regression and Multilevel/Hierarchical Models. Analytical Methods for Social Research, Cambridge University Press (2006)

[5] Gortney, M.E., Harris, P.E., Cerny, T., Maruf, A.A., Bures, M., Taibi, D., Tisnovsky, P.: Visualizing Microservice Architecture in the Dynamic Perspective: A Systematic Mapping Study. IEEE Access **10**, 119999–120012 (2022). `https://doi.org/10.1109/ACCESS.2022.3221130`

[6] Kolb, D.A.: Experiential learning: Experience as the source of learning and development. FT press (2014)

[7] Li, B., Peng, X., Xiang, Q., Wang, H., Xie, T., Sun, J., Liu, X.: Enjoy your observability: an industrial survey of microservice tracing and analysis. Empirical Software Engineering **27**, 1–28 (2022). `https://doi.org/10.1007/s10664-021-10063-9`

[8] McElreath, R.: Statistical rethinking: A Bayesian course with examples in R and Stan. CRC press (2020)

[9] Staron, M.: Action Research in Software Engineering. Springer Cham (2020). `https://doi.org/10.1007/978-3-030-32610-4`

[10] Sundelin, A.: epkanol/observability-tracing-experiential- learning: First formal release (Aug 2025). `https://doi.org/10.5281/zenodo.16790168`

[11] Vehtari, A., Gelman, A., Gabry, J.: Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. Statistics and computing **27**, 1413–1432 (2017). `https://doi.org/10.1007/s11222-016-9696-4`