What Data Scientists (Care to) Recall

Samar Saeed[1], Shahrzad Sheikholeslami[1], Jacob Krüger[2][0000−0002−0283−248X], and Regina Hebig[3][0000−0002−1459−2081]

[1] University of Gothenburg, Gothenburg, Sweden
{gussaesaa, gussheish}@student.gu.se
[2] Eindhoven University of Technology, Eindhoven, The Netherlands
j.kruger@tue.nl
[3] University of Rostock, Rostock, Germany
regina.hebig@uni-rostock.de

Abstract. To maintain and evolve a software system, developers need to gain new or recover lost knowledge about that system. Thus, program comprehension is a crucial activity in software development and maintenance processes. We know from previous work that developers prioritize what information they want to remember about a system based on the perceived importance of that information. However, AI-based software systems as a special case are not developed by software developers alone, but also by data scientists who deal with other concepts and have a different educational background than most developers. In this paper, we study what information data scientists (aim to) recall about their systems. For this purpose, we replicated our previous work by interviewing 11 data scientists, investigating the knowledge they consider important to remember, and whether they can remember parts of their systems correctly. Our results suggest that data scientists consider knowledge about the AI-project settings to be the most important to remember and that they perform best when remembering knowledge they consider important. Contrary to software developers, data scientists' self-assessments increase when reflecting on their systems. Our findings indicate similarities and differences between developers and data scientists that are important for managing the processes surrounding a system.

Keywords: Program Comprehension · Human Memory · Remembering · Data Scientists · Maintenance

1 Introduction

Development, maintenance, and evolution (we refer to *engineering*) processes surrounding software systems require the involved developers to obtain extensive knowledge about various properties. In particular, they must know details about the system and its engineering, such as the system architecture, the employed engineering processes, or the intended system behavior. Aiming to facilitate developers' work, researchers and practitioners continue to investigate and develop

techniques for recovering and documenting the respective knowledge. Such activities are key, but also expensive, within engineering processes. Past studies on these activities have focused on program comprehension and knowledge recovery, but little on whether, how, and what knowledge the involved developers aim to memorize [8]. Moreover, the focus has been on software developers and similar roles (e.g., testers, architects) that are closely connected to software development.

However, software systems are typically engineered by involving various different stakeholders and domain experts. Particularly with the rise of artificial intelligence (AI) and AI-based systems, data scientists have become more and more involved in engineering processes. Data scientists can have considerably different backgrounds and tasks compared to typical software developers [1,14,18,19], focusing on, for example, data collection, data cleansing, training AI models, or improving the performance of such a model. Unfortunately, research on AI-based systems focuses mainly on data quality, algorithms, performance, and similar technicalities, while the human aspects within the respective engineering processes have received less attention. In parallel, the findings of studies on developers' knowledge needs and what knowledge they consider important may not be fully transferable.

Overall, there are two gaps we aim to tackle in this paper: First, little research has focused on understanding what knowledge different stakeholder roles consider important, and thus aim to memorize. Second, none of these studies has focused on data scientists as a new, but more and more important, role. We address these gaps by replicating our previous interview survey on software developers' knowledge needs and memory performance [8] with 11 data scientists. So, in this paper, we investigate the information needs, memory, and perceived importance of knowledge of data scientists, contributing to a better understanding of their needs and characteristics in engineering processes. Our results can help researchers and practitioners alike, for instance, for developing novel techniques that are focused on helping data scientists with their specific AI-related knowledge needs.

2 Related Work

Developers need to continuously comprehend the system they are working on during an engineering process. This may include learning new things or recovering knowledge they have forgotten over time. Consequently, there has been extensive research on program comprehension, the activity of recovering knowledge about the source code a developer is working on [3, 15, 20, 21]. Research in this area investigates how developers comprehend code, what constructs (e.g., code comments [17], identifier names [4]) facilitate or complicate program comprehension, and proposes novel techniques for supporting developers. However, program comprehension is concerned with recovering detailed knowledge about the code and consequent system behavior, not tackling other knowledge issues like developers' memory or forgetting that we are concerned with. In our own previous works, we have been focusing on such issues, specifically developers' memory regarding source code [7, 11] and how to recover knowledge from different artifacts [10]. Most importantly, we [8] have previously investigated the relationship between

developers' information needs and their memory decay in a two-fold study. We collected questions that developers asked during their work by reviewing existing studies, classifying these questions into three themes: architecture, code, and meta. Then, we conducted a qualitative interview survey with 17 experienced developers working mostly on smaller systems, asking questions about their systems based on the three themes. We aimed to understand what knowledge developers consider important to recall from memory, assess their actual ability to remember such knowledge, and understand how they assess themselves in terms of memorizing. The results of our study imply that developers working on smaller systems tend to consider architecture and abstract knowledge about the code more important to remember, with meta knowledge being considered the least important to remember. For this paper, we replicated the interview part of our study with a different population: data scientists. Recently, in another study building on that previous work, we [9] have conducted a questionnaire to understand what knowledge developers consider important to memorize or document. So, we contributed complementary insights into how developers would prefer to document knowledge they may forget over time.

Differentiating between typical software developers and data scientists is important, since engineering AI-based systems exhibits different processes and consequent knowledge. For instance, Liu [14] performed interviews in which they identified 25 tasks covering various engineering phases that must be added to processes for AI-based systems. As a result, the types of knowledge that are relevant, their perceived importance to know from memory, and data scientists' ability to actually remember these may deviate compared to typical software developers—highlighting an important research gap. This gap has not been tackled in previous work on engineering AI-based systems. However, several other studies on such systems showcase the differences in terms of engineering processes [1], reasoning about or explaining AI-based systems [16, 23], or guiding end users [22]. Lastly, researchers have investigated the challenges for data scientists in engineering AI-based systems [18, 19]. All these works highlight the differences between data scientists and software developers, but do not investigate their knowledge needs and memory, which we study in this paper.

3 Methodology

For this paper, we replicated the interview part of our previous work [8] and built on systematic survey procedures [13]. Due to the different domains (software developers in the original versus data scientists in this study), we implemented some changes in the design that we explain in this section.

3.1 Research Questions (RQs)

With our study, we aimed to elicit what types of knowledge data scientists consider important to remember about their system ($\mathbf{RQ_1}$), how well they perform at remembering these ($\mathbf{RQ_2}$), and how their self-assessment matches their actual ability to remember knowledge ($\mathbf{RQ_3}$). For this purpose, we defined four RQs:

RQ$_1$ What knowledge about their system do data scientists consider important to memorize and remember?

RQ$_2$ Can data scientists correctly answer questions about their system based on their memory?

RQ$_3$ To what extent does a data scientist's self-assessment of their familiarity with a system align with their actual knowledge about that system?

RQ$_4$ What are the similarities and differences between data scientists and software developers?

We adapted the first three RQs from our previous study, and defined **RQ$_4$** to compare between both studies.

3.2 Interview Instrumentation

Identically to our previous study, we performed face-to-face interviews administered by the interviewer. An interviewer-administered interview limits the risk of misunderstandings and allowed us to gather more reliable data compared to a questionnaire. For our interview guide, we aimed to keep or adopt the questions from our previous study, since (1) these questions were established through a systematic literature review and (2) this allows a comparison of the outcomes for data scientists to those for software developers. We adjusted and added questions to account for the different artifacts data scientists work with, for which we built on the main practices of data scientists described by Burkov [2].

Questions. We provide an overview of the questions we asked during our structured interviews in Table 1. As we show, the questions align to five sections:

Overall Self-Assessment (OS): This section involves four questions on the interviewees' self-assessments regarding their memorized knowledge. These questions are identical to our previous study and we asked them four times: once at the beginning and once after each knowledge section (A, M, C). We did this to identify whether an interviewee's self-assessment would change after reflecting on their system.

AI-Project Setting (A): With these seven questions, we investigated our interviewees' memory of the project setting used for the AI-based part of their system. This section did not exist in our previous work and substitutes the questions about architecture that we asked the software developers.

Meta Knowledge (M): Next, we asked five questions on collaboration and system evolution, adapting one question (M_5) to the specifics of AI engineering.

Code Knowledge (C): In this last knowledge section, we asked four questions on code and AI-modeling details. Particularly, we adapted two questions (C_3, C_4) to AI engineering. Note that we adapted C_3 depending on the learning algorithm our interviewees used within their systems, differentiating between the types (e.g., supervised, unsupervised) and changing the respective details we asked for (e.g., labels, output).

Importance of Knowledge (IK): To wrap up, we asked the same five questions on the perceived importance of knowledge as in our previous study. In contrast to our previous study, we did not ask our interviewees to report their perceived importance of each individual question, but to only mention

Table 1. Overview of our interview questions. Those questions fully unchanged compared to our previous study [8] are asterisked (*).

id	questions and answering options (AOs)
	section: overall self-assessment (repeated after sections A, M, and C)
OS_1	* How well do you still know your system?
OS_2	* How well do you still know the architecture of your system?
OS_3	* How well do you know your code in the system?
OS_4	* How well do you know file *<three>*?
	. AO *<for each OS_i>*: 0–100 %
	section: AI-project setting (A)
A_1	What are the learning algorithms used in your system?
	AO: free text
A_2	Where did you get your data from?
	AO: ∘ provided by customer ∘ generated from another algorithm ∘ others (free text)
A_3	Did you continue collecting data? If yes, did you retrain your model?
A_4	Did you use a validation set or test sets? How did you split your training dataset?
A_5	Did you apply any feature engineering to your dataset? If yes, what techniques did you use?
A_6	Did you do any hyperparameter tuning? If yes, what techniques did you use?
A_7	Did you combine different models in your system? If yes, what techniques did you use?
	AO *<for each A_{3-7}>*: ∘ yes (free text) ∘ no
	section: meta (M)
M_1	* Can you point out an old file that has especially rarely/often been changed?
	∘ Yes (free text for file name) ∘ No
M_2	* How old is this file in the project life-cycle and how often has it been changed since its creation?
M_3	* Who is the owner of file *<one>*?
M_4	* How big is file *<two>*?
	AO *<for each M_{2-4}>*: free text
M_5	Did your model have any overfitting or underfitting? If yes, how did you fix it?
	∘ Yes (free text) ∘ No
	section: code (C)
C_1	* What is the intent of the code in file *<three>* and *<four>*?
C_2	* Is there a code smell in the code of file *<three>* or *<four>*?
	AO *<for each file in C_{1-2}>*: free text
C_3	If you use *<learning>*, can you describe your *<X>*?
	AO: free text *<for each>*
	<supervised learning>: *<feature vector & labels>*
	<semi-supervised learning>: *<feature vector & labels & data mostly labeled/unlabeled*
	<unsupervised learning>: *<feature vector & types of output>*
	<reinforcement learning>: *<feature vector>*
C_4	How did you assess your model's performance? What techniques did you use?
	AO: free text
	section: importance of knowledge
IK_1	* Which part of your system do you consider important?
	AO: free text
IK_2	* Which type of the previously investigated types of knowledge do you consider important?
	AO: ∘ architecture ∘ meta ∘ code
IK_3	Which of the previous questions do you consider important or irrelevant when talking about familiarity?
	AO: free text
IK_4	* What do you consider/reflect about when making a self-assessment of your familiarity?
IK_5	* Do you have additional remarks?
	AO *<for each file in IK_{4-5}>*: free text

those questions they considered particularly important or relevant. Our main motivation for this adaptation was to limit the time needed for each interview.

The first and last sections ask for opinions and experiences, while the other three (knowledge) sections ask for details about the systems we could verify afterwards.

Data-Science Adaptations. Initially, we planned to use all of our previous questions and to only add data-scientist-specific ones. However, this drastically increased the time needed for each interview (≈2 hours). Based on our test runs and potential interviewees raising the issue that this would be too long and not focused on their actual tasks, we decided to refocus the interviews on AI engineering.

For this reason, we exchanged the section on software architecture from our previous study with the section on AI-project settings, namely on the AI algorithms used (A_1), data sources (A_2), and similar strategical decisions relevant for engineering an AI. Note that we initially called this section "architecture" during the interviews as well. However, as several interviewees pointed out, this name was not the best fit to describe the type of questions we asked.

Similarly, we exchanged questions in the section code knowledge to refocus it from code details (e.g., return types, exceptions) towards the coding details of an AI. Specifically, we added a question about feature vectors and labels (C_3) as well as a question about the assessment of model performance (C_4). We also reduced the number of files about which we asked such questions from three to two to save time. In the section on meta knowledge, we removed the questions about the last changes to a file and instead added a question on whether there had been overfitting/underfitting and how this has been fixed (M_5).

Finally, we initially wanted to assess the importance of knowledge as we did before, namely for each individual question. However, to reduce the interview length, we decided to assess the importance of the knowledge sections only, and to depend on the qualitative analysis results that we obtained from the answers to IK_3. In the end, we limited the time of each interview to around 1 hour. Of course, this means that our new study is not an exact replication, but the focus on the specifics of AI engineering also promises more important insights into data scientists' knowledge needs.

Evaluation. Due to our adaptations, we needed to verify that our interview survey remained valid and reliable. Besides test runs and checks among ourselves, we asked particularly our first interviewee to give us their opinion about the questions and if there were any irrelevant or difficult-to-follow questions. According to that first interviewee, all questions made sense in the context of AI-based systems and did not need to be changed. We continued to ask all of our interviewees about the quality of the questions to reflect on potential threats to validity (part of IK_5). None of our interviewees indicated that our questions were hard to comprehend or irrelevant for data scientists.

3.3 Interview Conduct

Initially, we planned to conduct the interviews in person. However, most of our potential interviewees preferred online sessions, we were restricted in traveling due to the COVID-19 pandemic, and we ended up interviewing data scientists in different countries. For such reasons, we decided to conduct the interviews via online meetings using Zoom. This resulted in the structure and assessment of each interview as we describe in the following.

Interview Structure. We started each interview by introducing the interview protocol, checking for the interviewee's consent, and asking some background questions (cf. Table 2). Since most of our interviewees' systems were closed-source, we could not investigate those beforehand to prepare our questions (i.e., questions asking about specific files). To solve this problem, we asked each interviewee to write down four code files in the beginning, to which we then referred to in the respective questions (cf. Table 1). We aimed to avoid potential threats in the file selection by asking each interviewee to open their system and navigate to a file based on our suggestions. Specifically, we asked how many folders their repository involved and then randomly picked a number to select a folder, and we repeated the same for the files (or folders) in that folder. Using this method, we aimed to avoid our interviewees selecting a file they were particularly familiar with. Note that we ensured that the interviewee had worked on the file to ensure that we were not asking questions about a completely unknown piece of the system. Then, we continued with iterating through our questions without the interviewee looking into their system.
Rating Correctness. As in our previous study, we aimed to understand to what extent our interviewees could answer our questions correctly based on their memory. Unfortunately, we were not allowed to investigate the systems together with the interviewees, since the systems were closed source. For this reason, each interviewee had to re-iterate through the questions and assess themselves whether the question was correct or not—this time being allowed to look at the system's code and artifacts to compare their answers against the actual implementation.

While this strategy may have introduced bias, we trusted our interviewees to provide truthful assessments, since there were no negative consequences and they could check to what extent they could trust their memory. Moreover, while some questions can be easily self-corrected (e.g., $M_{1–4}$), others require a detailed reflection about various parts of the system and involve subjective opinions anyway (e.g., C_1, $A_{2–4}$). This problem is inherent to program comprehension, since two individuals may have different perceptions about the same concept. To mitigate potential threats, we asked the interviewees to explain their reasoning for each assessment to us, which we considered to improve the fairness. For instance, to compare their answers to C_1 and C_2 against the actual code, two interviewees

Table 2. Overview of the 11 included interviewees.

id	degree	exp.	domain	devs.
I_1	master	<1	finance	7
I_2	bachelor	15	telecommunications	3
I_3	master	2	machine learning	15
I_4	master	4	finance	6
I_5	master	10	healthcare	3
I_6	master	5	agriculture	4
I_7	PhD	5	software engineering	1
I_8	master	10	image processing	2
I_9	master	5	image processing	2
I_{10}	master	7	real estate	2
I_{11}	master	4	biomedicine	1

exp.: years of experience
devs.: number of developers involved

explained to us in detail what they thought the intention of each file was after looking at it again and what they considered to represent a code smell within each file. Note that we followed a similar strategy as in our previous study to deal with such inherently more subjective questions, which we found to yield reliable results and to connect better to an interviewee's memorized knowledge (which is based on their subjective perception, too).

Rating Scheme. To make it possible that we could compare the findings of our previous to this study, we employed the same rating scheme. Specifically, we awarded 0 points for incorrect, 0.5 points for partially correct, and 1 point for correct answers. We considered an answer partially correct if we and the interviewee noticed that an answer was missing important details that were relevant to the question. Identically, we awarded half a point if an interviewee was not completely sure about or confident in their answer.

3.4 Target Population and Sampling

We characterized our target population based on characteristics recommended in established guidelines [5, 13]. Namely, we targeted data scientists, initially those located in Sweden or working for Swedish companies. We did not put limitations on the educational level, gender, age, or years of experience in data science when recruiting interviewees. Due to time constraints, a lack of direct contacts, and other restrictions (e.g., COVID-19), finding interviewees within Sweden was challenging. Therefore, we decided to interview any data scientist who accepted our interview request, regardless of whether they worked in Sweden or for Swedish companies. In the end, we interviewed 12 data scientists of different genders, with varying years of experience, and working in various countries (e.g., Sweden, Germany) as well as domains. However, similar to our previous study, most of them have been working on smaller systems. Note that we excluded one interview during a quality check. The interviewee worked on a system at a very early development stage. As a consequence, seven out of 18 questions in the knowledge sections were not applicable and we could not assess the respective correctness. To improve the trust into our results, we decided not to use the respective interview data for our analysis. We provide an overview of the included interviews in Table 2.

We used judgment and snowballing sampling to recruit interviewees. Both are non-probabilistic sampling strategies, which we chose because we assume that the target population is rather large but its actual size and the respective individuals are unknown to us—which is why we cannot employ probabilistic sampling [13]. Specifically, we employed judgment sampling by contacting data scientists that we identified through searching on public company websites and Linked-In profiles. On Linked-In, we searched for data scientists who are working in Sweden. Then, we contacted the data scientists we could find via email and asked them to introduce us to any other data scientists that they thought may be interested to participate (i.e., snowballing). We also used our personal networks (convenience sampling), which yielded four additional interviewees from different countries.

3.5 Data Analysis

To answer **RQ$_1$**, we analyzed the interviewees' responses to the questions within the section importance of knowledge qualitatively and quantitatively. We used open coding on the free texts and computed the number of times each knowledge section was chosen as important or not important. To answer **RQ$_2$**, we quantitatively analyzed the correctness of the answers to the three knowledge sections. Specifically, we computed the average of the overall correctness for each question within the knowledge sections in Table 1 individually as well as for the knowledge sections combined. To answer **RQ$_3$**, we compared the interviewees' self-assessments to their correctness. Finally, to answer **RQ$_4$**, we compared our previous findings on software developers to those we obtained during this study for data scientists.

4 Results

Next, we present the results for each of our RQs individually.

4.1 RQ$_1$: Importance of Memorizing

To answer **RQ$_1$**, we asked our interviewees to choose the knowledge sections they think are important (I_2 in Table 1) while reflecting on the questions that we had in the questionnaire. Not surprisingly, considering their background, 10 out of the 11 interviewees chose AI-project setting knowledge as an important knowledge type, and seven out of the 11 interviewees chose code knowledge. Only one out of the 11 interviewees chose meta knowledge (cf. Figure 1). This interviewee selected all three types of knowledge as important.

To obtain more detailed insights, we also analyzed the answers to question I_3 to understand what questions we asked target knowledge that the interviewees consider important or irrelevant for reflecting on their familiarity with a system. Note that the interviewees chose what questions they wanted to make statements

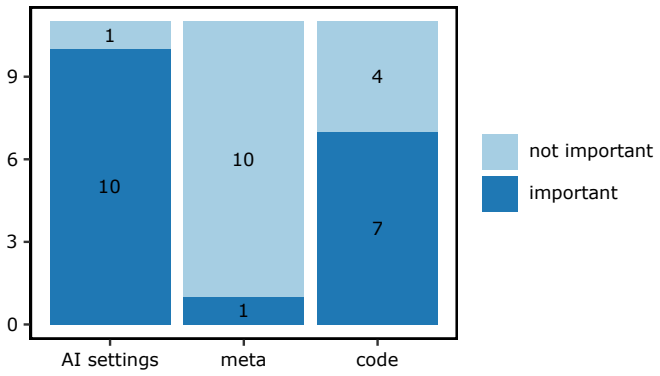

Fig. 1. IK$_2$: Importance of types of knowledge.

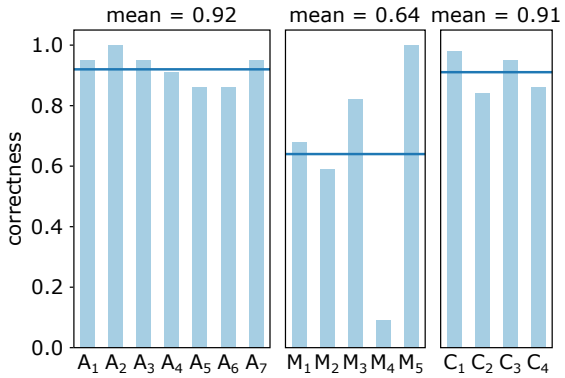

Fig. 2. Assessed correctness of our interviewees for each question on average.

about. AI-project setting questions were mentioned by six of the interviewees as being important when talking about familiarity. Most of them mentioned all questions in the AI-project setting section. One specified in more detail that they considered knowledge about the learning algorithms (A_1), data collection (A_3), feature engineering (A_5), and hyperparameter tuning (A_6) as important.

When it comes to code knowledge, three interviewees elaborated about what knowledge they consider important. One of them made a general statement that code is important for detailed knowledge. The second interviewee said that knowledge about code smells (C_2) and feature vectors/labels (C_3) is important. Lastly, the third interviewee only mentioned that feature vectors/labels (C_3) are important. Interestingly, another interviewee referred to knowledge about code smells (C_2) explicitly as not important. Lastly, six interviewees chose meta questions as not important, stating that questions about, for instance, the size of a file (M_4) are irrelevant to remember. Another interviewee mentioned explicitly that remembering whether an old file has been rarely or frequently changed (M_1) is not important.

Observations RQ_1 • The majority of our interviewees believes that AI-project setting knowledge is important to remember. • A little more than half of the interviewees thinks that code knowledge is important to remember. • The majority of the interviewees mentioned meta knowledge as not important when talking about familiarity with a system. •

4.2 RQ_2: Correctness of Memory

To assess how well our interviewees could remember the different parts of their systems (**RQ_2**), we calculated the average correctness of each question. We display a summary of the results in Figure 2.

AI-Project Settings. Most interviewees have scored noticeably high in the AI-project settings section. As we can see in Figure 2, the average correctness

for the AI-project settings section is 92 %. The highest average correctness is actually at 100 % and it was scored for A_2, which asks about the source of the data used in the system. In contrast, the lowest average correctness is 86 % and it was scored for two questions: A5, which asks about feature engineering, and A6, which asks about hyper-parameter tuning.

Meta. Our interviewees had the lowest correctness when it comes to meta knowledge. We can see in Figure 2 that the average correctness of the meta section is 64 %. However, our interviewees scored 100 % on question M_5 about whether the model in the system had an overfitting or underfitting. For question M_4, about the size of a file in terms of lines of code (approximated), our interviewees had an average correctness of only 9 %.

Code. Similar to the AI-project settings, our interviewees scored a considerably high correctness of 92 % for code knowledge. They scored on average 98 % correctly for question C_1 about the intent of two different files in the system. The lowest average correctness score is 84 % for question C_2 about the presence of code smells in selected files.

Overall Average. On average, our 11 interviewees reached a correctness score of 83 %, with one interviewee scoring a correctness of 100 % and the interviewee with the lowest correctness scoring 66 %.

Observations RQ_2 • Only one interviewee could remember all details we asked about correctly. • All interviewees could remember the source of their data and whether their model had an overfitting or underfitting. • Interviewees seem to remember the intent of their files, as well as their feature vector and labels better than code smells and the model-performance assessment techniques used. • Interviewees remember AI-project setting knowledge and code knowledge equally well. •

4.3 RQ_3: Self-Assessments Versus Correctness

To answer $\mathbf{RQ_3}$, we compared the interviewees' average correctness to their initial and final self-assessments, which we display in Figure 3. We used Kendall's τ [6] on this data to test for rank correlations. The results indicate no significant correlation between the overall self-assessment and the average correctness of the interviewees (both p-values > 0.05, initial $\tau = 0.242$, final $\tau = 0.061$). We also looked into how the results of the initial and final overall self-assessments changed during the interview. In particular, five interviewees increased and three interviewees decreased their overall self-assessment during the interview. Three interviewees left their self-assessments unchanged during the interview.

When analyzing the interviewees' responses to IK_4, we found that many of them reflect on their AI-project settings or architectural knowledge when making a self-assessment of how well they know their system. Namely, six of the interviewees described that they reflected on the pipeline flow and model training, their ability to explain the system structure or architecture, and the idea of the implementation. Three of these six interviewees also talked about reflecting on

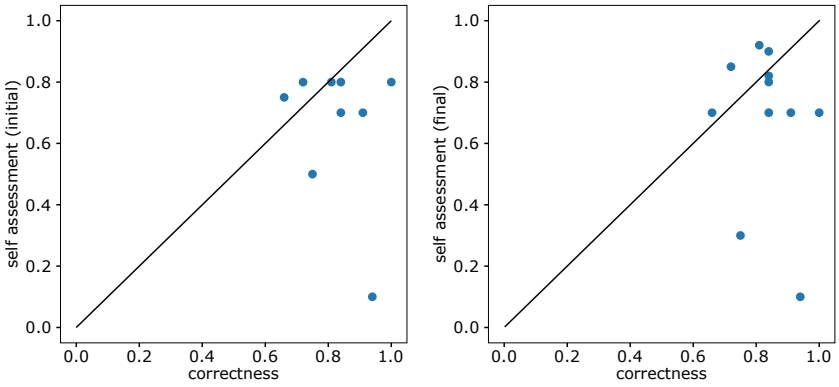

Fig. 3. Participants' self assessment in the beginning (left, some points overlap) and end (right) compared to their correctness. The diagonals serve as indicators only.

code knowledge. Other reflections were on what aspects are relevant in a system (two interviewees) and on the confidence in the own memory regarding the system (one interviewee). Lastly, one of the interviewees mentioned that they were not considering anything in particular, because the system was small and fairly recent and they were the only developer.

Observations RQ$_3$ • There is no correlation between interviewees' self-assessment of their familiarity and the correctness of their answers from memory. • The most mentioned aspect interviewees consider when making a self-assessment is their AI-project settings knowledge. •

4.4 RQ$_4$: Data Scientists Versus Software Developers

In Table 3, we compare the perceived importance that interviewees assigned to the different types of knowledge in this study (data scientists) and in our previous one [8] (software developers). We found that the results from both studies are similar with regard to the following aspects: The perceived importance is similar. AI-project settings knowledge is, identically to architectural knowledge, considered most important. Note that data scientists assign even higher importance to AI-project settings knowledge than software engineers do to architectural knowledge. The importance that is assigned to code knowledge and meta knowledge is comparable for both populations, too. We can observe that in both cases abstract knowledge, such as the intent of code [12], was considered more important. Also, both populations considered meta-knowledge to be the least important.

Observations RQ$_4$ • The order of perceived importance of knowledge types by data scientists and software engineers is similar: AI-project settings knowledge and architecture knowledge are followed by code knowledge and, lastly, meta-knowledge. •

Table 3. Comparison of the perceived importance of different types of knowledge.

type of knowledge	software developers [8]	data scientists
architecture	76.5 %	n.a.
ai-project settings	n.a.	90.9 %
code	52.9 %	63.6 %
meta	11.8 %	9.1 %

5 Discussion

In the following, we summarize our findings' implications for researchers and practitioners, before discussing threats to the validity of our results.

5.1 Implications for Researchers

Be careful with using self-assessments of familiarity. Just as for software developers [8], researchers should be careful when using data scientists' self-assessments of their familiarity with a system. Our results indicate no correlation between self-assessments and the actual correctness of answers.

Plan comparisons between data scientists and software developers carefully. Comparing between data scientists and software developers needs to account for differences in the way of thinking and working. Specifically, we formulate the following conjectures that should be investigated in future work:

- *A different intensity of knowledge use?* We observed that data scientists are specifically good at remembering answers to questions that are data-science specific (answering each question added to this study compared to our previous work with >80 % correctness). In contrast, they answered only one of the questions we also asked to developers with this high level of correctness. Thus, it does not seem reasonable to assume that they are simply better at remembering things. Another logical interpretation is that data scientists use individual pieces of knowledge about the data science-specific questions more often during their daily work. In contrast, software developers may have to switch contexts more often, recovering the same knowledge less frequently.
- *A different way to structure knowledge?* Data scientists seem to structure their knowledge about a system less in terms of files than software engineers would. This is supported by the observation that questions on meta knowledge that concern files ($M_{1–4}$) were answered with lower correctness by data scientists than by software developers.
- *A different notion of "big picture?"* In our previous study [8], software developers have indicated that the architectural knowledge is considered the big picture of a software system, with all interviewees agreeing on the importance of being able to draw a high-level architecture of the system. Even though we did not explicitly ask about architecture in this study, we collected some input from interviewees that indicate a different idea about what the "big picture" of an AI-enabled system is. One of the interviewees remarked that they are not too fond of software engineers who are "*always thinking about code,*" and

that they—as a data scientist—prefer to always think about design and the solution on a "higher level." We can only guess whether this higher level refers to AI-project settings knowledge or to a different type of knowledge that we did not capture here.

In future work, we need to investigate such questions in more detail.

5.2 Implications for Practitioners

Focus on AI-project settings during documenting and onboarding. Knowledge related to AI-project settings received the highest vote of importance, even compared to the importance software developers put on architectural knowledge. Thus, the respective pieces of information seem to be crucial for data scientists to perform their work, which is confirmed by their very high ability to answer these questions correctly without looking at their system (i.e., from memory). In practice, documenting an AI-system and onboarding new data scientists to an existing project should focus on these questions.

5.3 Threats to Validity

Internal Validity. The individual characteristics of our interviewees like their age, gender, or memory performance may have impacted their perceptions and responses. Due to the limited number of interviewees, we could not perform population analyses, which are subject to future work. Also, there is the risk that some questions may have been misunderstood by an interviewee. To mitigate this risk, we performed test runs among ourselves and at least one interviewer was present in each interview to explain questions and clarify potential confusions.

As we have mentioned before, we added and modified some questions to focus on knowledge relevant to data scientists. These new questions have a weaker foundation in empirical evidence compared to our original study for which we elicited the questions via a systematic literature review. However, to the best of our knowledge, there is currently no comparable data basis that we could have built on for data scientists. To still mitigate this threat, we derived our questions from the main practices of AI-based systems that are described by Burkov [2].

While we did not inform the interviewees in our previous study about the interviews' exact purpose to avoid biases, we had to briefly explain our motivation for this study to gain the interest of the data scientists. This may have caused a threat to the internal validity, because the interviewees may have prepared for our interviews by investigating their system beforehand or proposing to work on a system they know particularly well. While this is unlikely since all of them are working in practice, we further aimed to mitigate this threat. For this purpose, we did only share a short motivation for our interviews without details on the types of questions we would ask. Moreover, we asked rather different questions and chose files at random. On a related matter, there is the risk that an interviewee looked at their system while answering our questions, since the online setting limited our control at that point. However, this is very unlikely and we did not observe any behaviors that would hint at such a case.

As indicated in Section 3, we used the term "architecture knowledge" for the AI-project settings section during the interviews. Some interviewees pointed out that the term "architecture" is not the best description for that type of knowledge. In the end, we renamed the section for this paper using a term that was suggested by one of the interviewees: "project settings." It is possible that the use of the term architecture confused some interviewees and influenced their judgment of the importance of knowledge. We would assume that such confusion would decrease their rating of importance, and thus the values we present may be an underestimate. Moreover, among us authors, we discussed whether the placement of question C_4 (assessment of the model's performance) in the code knowledge section is appropriate, or whether the question should have been placed in the AI-project settings section. It is possible that the placement changed the perceived importance of the code knowledge section.

External Validity. The primary external threat to this study is the small sample size. We analyzed only 11 interviews, which means that it is not clear to what degree the results can be generalized. This uncertainty holds especially with regard to data scientists working on larger systems, since we mostly studied data scientists working on smaller systems. Furthermore, after interviewing data scientists who are working in different companies, it became clear that the tasks assigned to data scientists in such companies vary. This may cause individuals with the same title (data scientist) to have very different responsibilities, which eventually leads to varying perceptions. Such threats are partly inherent to interview surveys as well as human cognition, and thus we cannot fully overcome them. Still, we aimed to mitigate such threats by involving a diverse sample of data scientists to cover broader experiences and perceptions.

6 Conclusion

In this paper, we presented an interview survey with 11 data scientists, investigating what knowledge they consider important, aim to remember, and how they perform when recalling it from memory. For this purpose, we replicated parts of our previous work with software developers [8], against which we also compared our results. Among others, our results indicate that data scientists (identical to software developers) consider more abstract knowledge more important to remember, and perform quite well at recalling it from memory. Also, we highlighted differences between data scientists and software developers, particularly with respect to what constitutes "more abstract" knowledge. Based on these findings, we have discussed concrete recommendations and steps for future work to improve our understanding of data scientists' knowledge needs and to facilitate their work.

References

1. Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., Nagappan, N., Nushi, B., Zimmermann, T.: Software engineering for machine learning: A case study. In: ICSE-SEIP. IEEE (2019)

2. Burkov, A.: The hundred-page machine learning book. Andriy Burkov (2019)
3. Haiduc, S., Aponte, J., Marcus, A.: Supporting program comprehension with source code summarization. In: ICSE. ACM (2010)
4. Hofmeister, J., Siegmund, J., Holt, D.V.: Shorter identifier names take longer to comprehend. In: SANER. IEEE (2017)
5. Kasunic, M.: Designing an effective survey. Tech. Rep. CMU/SEI-2005-HB-004, Carnegie Mellon University (2005)
6. Kendall, M.G.: A new measure of rank correlation. Biometrika **30**(1/2) (1938)
7. Krüger, J., Çalıklı, G., Berger, T., Leich, T.: How explicit feature traces did not impact developers' memory. In: SANER. IEEE (2021)
8. Krüger, J., Hebig, R.: What developers (care to) recall: An interview survey on smaller systems. In: ICSME. IEEE (2020)
9. Krüger, J., Hebig, R.: To memorize or to document: A survey of developers' views on knowledge availability. In: PROFES. Springer (2023)
10. Krüger, J., Mukelabai, M., Gu, W., Shen, H., Hebig, R., Berger, T.: Where is my feature and what is it about? a case study on recovering feature facets. Journal of Systems and Software **152** (2019)
11. Krüger, J., Wiemann, J., Fenske, W., Saake, G., Leich, T.: Do you remember this source code? In: ICSE. ACM (2018)
12. Krüger, J., Li, Y., Zhu, C., Chechik, M., Berger, T., Rubin, J.: A vision on intentions in software engineering. In: ESEC/FSE. ACM (2023)
13. Linåker, J., Sulaman, S.M., Höst, M., de Mello, R.M.: Guidelines for conducting surveys in software engineering. Tech. rep., Lund University (2015)
14. Liu, H., Eksmo, S., Risberg, J., Hebig, R.: Emerging and changing tasks in the development process for machine learning systems. In: ICSSP. ACM (2020)
15. Maalej, W., Tiarks, R., Roehm, T., Koschke, R.: On the comprehension of program comprehension. ACM Transactions on Software Engineering and Methodology **23**(4) (2014)
16. Montavon, G., Samek, W., Müller, K.R.: Methods for interpreting and understanding deep neural networks. Digital Signal Processing **73** (2018)
17. Nielebock, S., Krolikowski, D., Krüger, J., Leich, T., Ortmeier, F.: Commenting source code: Is it worth it for small programming tasks? Empirical Software Engineering **24** (2019)
18. Pereira, P.: Towards helping data scientists. In: VL/HCC. IEEE (2020)
19. Pereira, P., Cunha, J., Fernandes, J.P.: On understanding data scientists. In: VL/HCC. IEEE (2020)
20. Roehm, T., Tiarks, R., Koschke, R., Maalej, W.: How do professional developers comprehend software? In: ICSE. IEEE (2012)
21. Schröter, I., Krüger, J., Siegmund, J., Leich, T.: Comprehending studies on program comprehension. In: ICPC. IEEE (2017)
22. Winkler, J.P., Vogelsang, A.: "What does my classifier learn?" A visual approach to understanding natural language text classifiers. In: NLDB. Springer (2017)
23. Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., Lipson, H.: Understanding neural networks through deep visualization. arXiv (2015), arXiv:1506.06579