CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

What Data Scientists (care to) Recall

Bachelor of Science Thesis in Software Engineering and Management

Samar Saeed

Shahrzad Sheikholeslami

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022

CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

What kind of knowledge do data scientists consider important enough to remember about their (small) systems?

Supervisor: Regina Hebig
Examiner: Richard Berntsson Svensson

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

[Cover: an explanatory caption for the (possible) cover picture with page reference to detailed information in this essay.]

Department of Computer Science and Engineering
University of Gothenburg
Chalmers University of Technology
Gothenburg, Sweden 2022

What Data Scientists (care to) Recall

1st Samar Saeed
Computer Science and Engineering
University of Gothenburg
Gothenburg, Sweden
gussaesaa@student.gu.se

2nd Shahrzad Sheikholeslami
Computer Science and Engineering
University of Gothenburg
Gothenburg, Sweden
gussheish@student.gu.se

***Abstract*—Program comprehension is a crucial activity for software developers, just as it is for data scientists. It is an activity that involves gaining new knowledge and recovering lost knowledge, and the process could be a factor that affects various aspects of software projects. Because of this, there is a good amount of research on developers' information needs and program comprehension support tools and techniques. "What Developers (Care to) Recall" [1] especially investigates the link between what software developers think is important to remember, their information needs and their memory. Krüger et al. studied the importance of knowledge, memory correctness, and self-assessment by interviewing 17 developers of small systems. However, we could not find similar studies that particularly focus on data scientists and their human factors. Data scientists deal with different concepts in their daily tasks, which means that their information needs may be different from software developers'. To fill this gap, we replicated [1] and conducted the same interview-survey with some adjustments to the questions fit in the data science context. We interviewed 12 data scientists and investigated the knowledge they consider to be important to remember, whether they can remember parts of their systems correctly, the relation between their actual knowledge and their self-assessment, and finally how different/similar the results are to the replicated paper's. Our results suggest that similar to software developers, data scientists consider architectural knowledge to be the most important to remember, they perform best in what they considered to be the most important type of knowledge, and on the contrary to software developers, their self-assessment increases when reflecting on their systems. In this paper, we discuss these findings, as well as the validity of these results and what kind of research directions may need to be considered in the future to better grasp the kind of comprehension support that data scientists need.**

***Index Terms*—data scientists, human factors, program comprehension, knowledge importance, memory**

I. Introduction

Developers need to understand the behaviour and structure of the systems they work on in their daily work in order to maintain, develop and extend them, which makes the process of code comprehension one of the factors that may affect the success of software projects, maintenance costs, the quality of the system, and so forth [2] [4] [5] [6]. Therefore, researchers continue to investigate and suggest techniques that may help developers by facilitating tasks such as code comprehension and recovering lost knowledge. At the present time, developers spend around 58% of their overall time on comprehending source code, and this includes for example legacy system parts that they maintain [17]. Program comprehension is a discipline that is concerned with supporting developers in this crucial task.

To understand the approaches developers use when they comprehend their systems, as well as the types of knowledge they consider important to have when working on their tasks, researchers have investigated important aspects of a developer's daily tasks such as program comprehension [3]. For instance, Maleej et al. [3] have found that developers' information needs mostly lie within functionality knowledge (e.g. code intent), as well as rationale knowledge (e.g. questioning design), and that developers do not consider meta knowledge (e.g. who worked on what) to be important. The results of their study also shows that developers follow their own pragmatic procedures based on context during program comprehension, and are unaware of program comprehension tools and therefore did not use them. This may indicate that these tools perhaps did not help the developers who came across them enough to share them with other developers, as useful tools tend to get popular fairly quickly in the software development field.

Krüger et al. [1] have investigated the types of knowledge developers – who worked on smaller systems – considered important such as architecture, code, and meta knowledge, aiming to study the link between developers' memory and their information needs. Their results suggest that developers consider architecture the most important to remember, followed by code, while meta knowledge is considered the least important, akin to the results of [3]. However, these studies examined software developers of all types of software fields, which means that generalisations based on their results might not align well with data scientists, as their daily tasks can be quite different from the typical software developer [9] [18].

Data science can be quite complex, as data scientists constantly work with data, and deal with concepts that are special to data science. Their daily tasks require them to: *1) Collect needed data 2) Clean the data to be used in learning algorithms 3) Train the machine learning models 4) Estimate and improve the precision and accuracy of the trained models* [12]. To the best of our knowledge, there is a lack of studies that investigate the types of knowledge data scientists consider important to keep in their memory. Most data science research papers cover the technical areas, whereas questions on the human aspects and factors of data scientists are still not appropriately and adequately researched. There is research on the understandings of users of machine learning-

based systems, for example [7], but not the developers of the machine learning system themselves.

In this thesis, we want to take a look at data scientists' perspectives and investigate what they care about enough to remember about their systems by replicating and extending the original study "What developers (care to) recall: An Interview Survey on Smaller Systems" that investigated developers' memory in a larger context [1]. Therefore, the purpose of this study is to investigate the information needs and memory decay of data scientists to obtain conclusions that can potentially be beneficial for both researchers and practitioners in the data science field. For instance, the results may support other researchers in developing techniques that can aid data scientists in comprehension, which may also lead to introducing a framework that novice data scientists can refer to when they are dealing with systems that are new to them which they need to maintain and extend.

Moreover, the results may also contribute to devising and proposing techniques and tools that can make data scientists' daily tasks (e.g. program comprehension, knowledge recovery) smoother. Furthermore, we want to compare the results of this study to the original study which studied software developers in a bigger context, as we might be able to draw conclusions that can highlight the differences between a typical software developer's memory and a data scientist's memory based on practical evidence.

II. Related Works

Developer Memory and Comprehension of Systems from the perspective of the engineer

Developers need to continuously understand their systems in their daily work, which may include learning new things about their systems, or recovering details that they have forgotten about the systems; hence the large number of papers available which focus on developers' comprehension and memory.

For example, Maleej et al. investigated how developers comprehend programs in their daily work, as well as the types of knowledge developers consider important through qualitative and quantitative research [3]. They observed the comprehension behaviour, tools used, and strategies of 28 experienced developers, and designed an online survey with 1477 respondents. The survey had questions about the types of knowledge developers consider important for program comprehension, the source of these knowledge types, and how developers share them with their colleagues. The results of the research indicate that developers follow pragmatic context-dependent comprehension strategies, and that they did not make use of program comprehension facilitating tools, seemingly unaware of them. Maleej et al. conclude that there seems to be a gap between research and practice [3]. As mentioned earlier in the introduction, this result suggests that the existing tools for program comprehension support may not be as useful as developers would want them to be, and that further research on program comprehension is needed.

Consequently, research that suggests comprehension support tools continues. For instance, Haiduc et al. talk about the challenges and problems developers encounter during program comprehension, especially when the amount of code is vastly large and thus difficult to maintain and keep up with [13]. To make it easier and faster for developers to fathom source code and facilitate their comprehension, Haiduc et al. propose a solution which is to use high-level descriptions of the source code by automatically determining said descriptions, and auto-generating summaries using automated text summarization technology.

In [15], Roehm et al. did an observational study to find out what different developers do in order to comprehend and understand the software they are working on. They did their study on 28 developers from 7 different companies to test the previous observations mentioned by the researchers who worked on this subject previously and also report new observations. They claimed that the developers see the code comprehension as a part of maintenance tasks and not an individual task itself. Although comprehending the software you are working on is an important part of software development, developers usually try to skip this step. Their study results indicate that the developers are not familiar with the tools which are available for them to do the code comprehension as a separate task during development. Software comprehension is considered a complex task for developers. [16] stated that developers need to look into both the "structure of their source code" and the "core domain of the software" in order to be able to comprehend their software. That is why this task is not considered easy to do by developers. They looked into different methods that can be used to help the developers to understand their system. This shows that there are different methods that can be used for system comprehension and they seem to be feasible based on this study.

In order to understand the type of information developers need to recognize what might help in facilitating their program comprehension and knowledge recovery, researchers have investigated the relationship between the developer's information needs and their memory decay in a two-fold study. Following SLRs guidelines, Krüger and Hebig [1] collected questions that developers asked during work from different studies, classifying them based on 3 main themes; architecture, code, and meta. They conducted a qualitative interview survey with 17 experienced developers – working on smaller systems –, using general questions based on the 3 main themes regarding their systems, with the goal of understanding what developers consider important to recall, assessing their ability to remember knowledge about their systems, and finding out how the way they assess their knowledge relates to their actual knowledge. The results of their study imply that developers working on smaller systems tend to consider architecture and abstract knowledge about the code to be the most important to remember, with meta knowledge being the least important to remember. However, the study interviewed developers in a larger context, working on smaller systems, which means that the results may change if for instance, developers from the data science field or developers working on larger-scale systems were interviewed.

Human factors in Data Science

There is a lack of studies that investigate the human factors of data scientists. Data scientists such as machine learning developers work differently from the traditional software developer, therefore, the human aspect of data scientists demands attention and investigations. The majority of today's data science studies are focusing more on the technical side; as in how different learning algorithms and trained models work in real-life, how to define a process for implementing machine learning systems, what tasks and steps should be added to the traditional software development processes, and what are the challenges data scientists face during development. We looked into specific papers which are focusing on two main concepts: Human factors of software engineering in machine learning and Software Comprehension and memory of developers.

"Emerging and changing tasks in the development process for machine learning systems" claims that there is an existing gap in today's knowledge regarding the changes needed to be applied to the software development process when machine learning related components are added to the software. By doing an interview study, they were able to come up with 25 tasks which need to be added to the software development process. These tasks are done in different software development phases such as Requirement Engineering, Development Iterations, Deployment etc. 25% of their founded tasks are part of the Requirement Engineering phase which is the first phase in the software development process [8]. This shows the importance of having a data scientist from the beginning of the project. Also, we can see the presence of the data scientists in all of the development phases. This also emphasises the importance of investigating machine learning human factors in the later studies. In addition, the paper states that there needs to be some knowledge exchange between the team during the process [8], which shows that researching on the topics which machine learning engineers (data scientists) recall or believe that they are important to recall will be useful in developing machine learning based software.

A case study applied to the Microsoft teams [9], which are specifically working on AI features, also looked into the challenges that an AI specific software team may face during the development. Their nine stage workflow which is specific to the development of AI features shows that although there are some similarities between the development of AI-based software and other types of software, we need to come up with a process/workflow which is more specific to AI. They claimed that their key finding is three new domains which are specific to AI and will differentiate AI with other types of software, which are 1) More complex discovering, management and versioning the needed data 2) Different skills needed for model customization and reuse tasks 3) More difficulty in handling AI components due to the presence of entanglement between modules [9]. The results of this study shows that although AI-based software is a type of software, there are some complexities and difficulties in the process of their development which needed to be more emphasised on and looked into individually.

Comprehension of ML/AI-Models

As mentioned above, there are papers which focus on how model training works and are trying to explain them in an easy way to understand. Their goal is to help the users to understand that machine learning and artificial intelligence is not a magical black box. For example, "What does my classifier learn?" is a conference paper published in 2017 which explains how a trained neural network model is making decisions in the natural language processing (NLP) context [7]. [21] and [14] are also other examples of papers which are aiming for providing clarity in how neural networks work and what are the mathematical reasons behind the learning algorithms. Although there is research done on comprehension from the user's perspective of systems built by machine learning models [7], we were not able to find articles which focus on data scientists' comprehension of their systems.

On the other hand, there are research papers that aim to understand the role of data scientists to identify the challenges they face, as well as the way they work and make use of existing methodologies and tools in order to find solutions that could assist data scientists in their work [19] [20]. However, as far as we currently know, research on the information needs of data scientists, the knowledge data scientists perceive as "important", as well as how both the previous two points may be linked to their memory decay, has not been done yet.

III. Methodology

In order to have a systematic survey procedure, we decided to follow the steps that are mentioned in the "Guidelines for conducting surveys in Software Engineering" published in 2015 [10]. The paper stated that the survey methodology consists of two main parts which are *planning* and *applying the planned steps*. They are introducing 8 sequential steps which can be followed by researchers who choose surveys as their research method. We should state that we did not apply all the methods and techniques which are mentioned in this article. The important and useful steps which were accomplish-able based on the scope of our project were derived from the paper. Also, it is necessary to state that most of our research methods are the repetition of the methods mentioned in the paper we are replicating and extending [1]. There are some few changes applied to the previous study which will be explained in detail in the following sections.

Research Objectives

As mentioned in the previous sections of the paper, the purpose of this research is to find out what type of knowledge do data scientists remember about the system they are working on, and what concepts do they consider important to recall. The existence of a correlation between the concepts they can remember well, and the concepts which they consider important will be looked into as well [1]. Extending and replicating the previous research with changing the structure and the content of the used interview questionnaire, and

interviewing data scientists are the objectives of this research. We are also curious about how the results may be different from the replicated paper by applying the above changes. The research questions are equivalent to the ones in the original study [1], but with a change in the interviewees, as we will be targeting only data scientists instead of all types of developers.

- **RQ1**: What knowledge about their system do data scientists consider important to remember?
- **RQ2**: Can data scientists correctly answer questions about their system based on their memory?
- **RQ3**: To what extent does a data scientist's self-assessment align with their actual knowledge about their system?
- **RQ4**: What are the similarities and differences between data scientists and developers when it comes to what they consider to be important to remember?

Survey Instrument

We decided to use a face-to-face interviewer-administered questionnaire as our survey instrument [1] [10]. We believed that based on the replicated study [1], choosing this type of questionnaire was applicable. An interviewer-administered questionnaire will reduce the risk of misunderstanding, and will allow us to gather more reliable data. Our questionnaire consists of questions derived from the source paper (which is conducted via systematic literature review by Krüger et al.), and the questions added by us. Due to applied changes to the population (targeting only data scientists), a number of questions are added to the questionnaire to make the questions more specific and related to data science concepts. To gain a more accurate insight in data science, we tried to formulate these questions based on the main practices of data science mentioned in [12]. The questionnaire contains both open and close ended questions and is divided into 5 main sections:

- **Overall self-assessment:** This section contains questions to find out the interviewee's own opinion and impression on their assessment of the system. The interviewee must answer this section four times; one time in the beginning, and then once after each knowledge section is answered excluding the Importance of Knowledge section. This is to see if their self-assessment would change after answering each knowledge section.
- **Architecture:** This section is the first knowledge section and it consists of seven questions about the architecture and structure of the system.
- **Meta knowledge:** This section is the second knowledge section and it consists of five questions that ask about the context of the system.
- **Code comprehension:** This section is the third knowledge section and it consists of four questions (Two of them have sub-questions) that focus on the implementation of the system.
- **Importance of knowledge:** This section consists of five questions that ask about which part of the system and the knowledge we categorised the interviewee thinks is more important.

Architecture, Meta Knowledge, and Code Comprehension are the sections that require the interviewee to answer questions about their systems.

TABLE I
INTERVIEW QUESTIONS

ID	Questions
	Section: Overall Self-Assessment (0-100%)
OS1	How good do you still know this system?
OS2	How good do you still know the architecture of your overall system?
OS3	How well do you know your code in this project?
OS4	How well do you know File 3?
	Section: System Files
	File 1: File 2: File 3: File 4:
	Section: Architecture
A1	What are the learning algorithms used in your system?
A2	Where did you get your data from? () Provided by customer () Generated from another algorithm () Others (Please write down your answer)
A3	Did you continue collecting the data? If yes, did you retrain your model?
A4	Did you use a validation set or test sets? How did you split your training data-set?
A5	Did you apply any feature engineering to your data-set? If yes, what techniques did you use? () Yes () No — Technique:
A6	Did you do any hyperparameter tuning? If yes, what techniques did you use? () Yes () No — Technique:
A7	Did you combine any different models in your system? If yes, what techniques did you use? () Yes () No — Technique:
	Section: Meta Knowledge
M1	Can you point out an old file that has especially rarely/often been changed? () Yes () No — File:
M2	How old is this file in the project life-cycle and how often has it been changed since the creation?
M3	Who is the owner of File 1?
M4	How big is File 2?
M5	Did your model have any overfitting or underfitting? If yes, how did you fix it? () Yes () No — How:
	Section: Code Comprehension
C1	What is the intent of the code in the file? A. File 3: B. File 4:
C2	Is there a code smell in the code of the file? A. File 3: B. File 4:
C3	Feature vector and labels () Supervised Learning — () Can you describe your feature vector? — () Can you describe your labels? () Semi-Supervised Learning — () Can you describe your feature vector? — () Can you describe your labels? — () Which type of examples are the majority in the dataset? Labelled or unlabeled? () Unsupervised Learning — () Can you describe your feature vector? — () Can you describe the types of your output? () Reinforcement Learning — () Can you describe your feature vector?
C4	How did you assess your model's performance? What techniques did you use?
	Section: Importance of Knowledge
IK1	Which part of your system do you consider important?
IK2	Which type of the previously investigated types of knowledge do you consider important?
IK3	Which of the previous questions do you consider important or irrelevant when talking about familiarity? (Pick the questions)
IK4	What do you consider/reflect about when making a self-assessment of your familiarity?
IK5	Do you have additional remarks?

A. Instrument Evaluation

We kept adjusting the questions from the survey used in [1] and the new added questions according to the feedback given by the supervisor and applied many changes. For example, in the Architecture section, we removed some of the questions we thought were not relevant in the context of our research. We also removed questions that asked about database functionalities, user interface, and the system's main controller. Initially, the questionnaire had more questions since we planned to have interviews that can take up to 2 hours based on the time mentioned in the replicated study [1], but we received complaints from some of the potential interviewees we contacted about the interview duration being too long. Thus, we had to remove some of the questions to make the duration of the interview not exceed 1 hour.

To make sure that our survey instrument is valid and reliable, we asked our first participant to give us their opinion about the questionnaire and if there were any irrelevant or difficult to follow questions. According to the first participant, all the questions made sense and did not require any changes. We continued to ask all of our participants at the end of the interviews about the quality of the questions to keep the threats to validity in check. We have not received any feedback that suggests that our questions are not understandable or irrelevant in the data science context.

B. Interview Structure

Initially, we planned to conduct the interviews in person which we managed to do with the first interviewee. However, we noticed that the majority of the potential interviewees we contacted did not prefer to have the interview in person. Taking this into consideration, as well as other factors such as the spread of COVID-19, and potential interviewees not being available in Gothenburg for an in-person interview, we decided to switch to conducting online interviews via Zoom, despite the disadvantages and threats to validity.

We started the interviews by introducing the interview protocols, and then asking the interviewee to fill in the background questions, as well as sign a consent form in case of paper publication. The majority of our interviewees' projects were not open source which is why we had no chance to access and check their systems to fill in the questions that for example ask about certain files in their systems. In order to solve this problem, we devised a roundabout way to ask about files without looking at their systems. First, we replaced the empty spaces in some of the interview questions that required filenames with File 1, File 2, and etc. Second, we added an empty file list in the beginning of the questionnaire (Table I). Before answering the knowledge sections, the interviewees had to fill in the empty file list with filenames from their systems so that they could refer to them when answering the questions that contain File 1, 2, 3, 4 such as M3, M4, and C1 (Table I).

To fill in the file list, we asked the interviewees to open their systems and navigate according to our selection. For instance, we asked the interviewee about how many folders their repository had, and then we randomly picked a folder (e.g., the second folder). Then, we asked about how many files the folder we picked had, and randomly picked a file (e.g., the forth file) that they added to one of the files in the empty file list. After the file selection process, we asked the interviewees to start answering the questions in the knowledge sections without looking at the system, and we clarified any misinterpretations when needed throughout the questionnaire/interview.

C. Rating Correctness

The correctness of answers was rated by going through each question with the interviewees, asking them to check their systems to self-correct themselves. We were unable to access and investigate the interviewees' systems and assess correctness together with them as [1] did due to privacy issues since the majority of our interviewees' systems were closed source. As a result, we did not have a way to even evaluate the interviewees' self-correction so we could not alleviate potential bias. Some of the questions were easy to self-correct and had no place for bias such as M1-M4 (Table I) which the answers to them should be documented in version control, but answers to high-level questions such as A2-A4 (Table I) were difficult to self-correct by just looking at the system. Most of our interviewees self-corrected the architecture questions based how confident they were about their own understanding of the system structure.

Other questions such as C1-C2 (Table I) do not necessarily have one definitive correct answer to them, since they ask about the intent and code smells of files which are usually subjective and debatable topics among developers. Our interviewees self-corrected their answers to C1-C2 based on their own perspectives and understanding of the code. The interviewees however did reflect and explain the reasoning behind their self-corrections at times, as they were trying to be fair in their judgement as much as possible. For example, for questions C1-C2, two interviewees gave us percentages of how much their answers covered all intents and code smells that can be found in the selected files, and explained in detail to us what makes them for example think that something is a code smell, especially if the file had code written by one of their colleagues.

D. Rating Scheme

Since we decided to analyse our data in the same method as [1], we followed the same rating scheme which is 0 points for incorrect answers, 0.5 points for partially correct answers, and 1 point for correct answers. To rate whether an answer is partially correct or not, we asked the interviewees to let us know if their answers were missing important points and did not cover everything that should be mentioned in the answer. Additionally, If the interviewee was not completely sure or confident enough, we considered their answers to be partially correct.

Target Audience and Sampling

To answer our research questions, we targeted data scientists that are based in Sweden and working for Swedish companies and branches. We tried to characterise them based on attributes that are recommended in [10] [11].

- **Size:** We do not have access to the numbers of data scientists in Sweden but we are aware that our target audience is smaller than the replicated paper's target audience.
- **Jobs and Responsibilities:** Working as a data scientist.
- **Education Level:** No limitation.
- **Gender:** No limitation.
- **Age:** No limitation.
- **Years of Experience:** No limitation.

We tried to contact data scientists of different companies, domains, and experience but due to time constraints and our status as bachelor students, finding potential interviewees who would approve to get interviewed was difficult, so we decided to interview any data scientist who accepted our interview request, regardless of diversity. We mentioned above that we initially targeted data scientists working in Sweden only, but after switching to online interviews, we removed this limitation and we were willing to interview any data scientist from our network regardless of their location and the company they work for. We managed to interview 12 data scientists of different gender, years of experience, and domains but all working on smaller systems with the exception of one data scientist (ID3, Table II) who worked on a medium sized system.

TABLE II
OVERVIEW OF THE INTERVIEWEES

ID	Degree	Exp	Domain	Devs	Status
1	Master	< 1	Market domain prediction	7	Ongoing
2	Bachelor	15	Telecom	3	Done
3	Master	2	Machine Learning	15	Ongoing
4	Master	4	Credit / Risk	6	Ongoing
5	DSc	10	Healthcare / HR	3	Ongoing
6	Master	5	Agricultural classification	4	Done
7	PhD	5	Software Performance AI	1	Done
8	Master	10	Image Processing	2	Ongoing
9	Master	5	Image Processing	2	Ongoing
10	Master	7	Real Estate	2	Ongoing
11	Master	4	Biomedicine	1	Ongoing

We used *Judgement sampling* and *Snowballing sampling* to create and define our sample. Both Judgement and Snowballing sampling are a type of Non-probabilistic sampling. The reason for choosing non-probabilistic sampling methods is that the population is very big and we are not able to apply probabilistic sampling although we know that using non-probabilistic sampling would make our results less generalizable [10]. The Judgement sampling was applied by contacting specific employees by searching through company websites, and Linked-In. We filtered the search engine in Linked-In to look for data scientists who are working in Sweden. We contacted the data scientists we found via email and also asked them to introduce us to any other data scientists that they thought might be interested to participate which counts as a Snowballing sampling. We also utilised the network of our supervisor to find relevant participants, which gave us the opportunity to interview four data scientists from our sample.

Data Analysis

We analysed the data we collected in the same way done in the replicated study [1] but with very minor changes. This way, we could keep the results consistent enough and could compare the results of our study with the replicated paper to see how changing our population might affect the results of [1], which answers our fourth research question. In order to answer the first research question, we analysed the interviewees responses to the Importance of Knowledge section (Table I) qualitatively and quantitatively by extracting codes and calculating the number of times each knowledge section was chosen as important and not important. The quantitative analysis we made based on the correctness of the answers to the Architecture, Meta Knowledge and Code Comprehension sections (Table I) helped us in answering the second research question. For this analysis, we calculated the average of overall correctness for each question (A1-A7 — M1-M5 — C1-C4 in Table I), and also for each knowledge section (Architecture, Meta, Code). The quantitative analysis we made based on the self-assessment scores were used to answer the third research question. For this analysis, We calculated the p-value using the average correctness of each participant, the initial and final overall self-assessments using Kendall's τ.

Although we interviewed 12 participants, we decided to exclude one of our data points for being an outlier. One of our participants chose a system that was in the early development stage and because of this, 7 out of the 18 questions in the Knowledge Sections were not applicable to the system. Since these 7 questions could not be answered, we could not score the correctness for them. Therefore, our calculations come from 11 different data points only. We initially wanted to calculate the importance of knowledge as done in [1] but due to the interview length constraint, it was difficult to ask the interviewees to rate the importance of each question in the questionnaire. Thus, we decided to depend on the qualitative analysis results that we obtained from the answers to IK3 (Table I).

Threats to Validity

Drawing Conclusions: After analysing our data and before coming to conclusions, we needed to make sure that our chosen research procedure is valid, to make sure that our results will be reliable and generalizable under specific circumstances. The threats mentioned by [1] are the ones which affect the paper's Internal, External, and Conclusion validity. Due to the fact that our research is a replication of [1], we can assume that the whole threats mentioned in [1] are also a threat to our research validity. There might also be some more threats more specific to our research due to the changes we applied to the research procedure.

Internal Validity: We are not considering how human factors such as age, gender, memory performance, etc. can affect our study. The mentioned aspects are more related to the psychological side of the study which is not in the scope of our research and knowledge. The whole interview questions might not be an ideal decision and questions might be misunderstood by the interviewees. As it is mentioned in the methodology section, we added some questions to the questionnaire which are related to the data science field. Choosing these questions and grouping them into the pre-defined groups might not be an ideal choice.

In addition, unlike in [1], we could not avoid explaining briefly the purpose of the interview to the potential interviewees to get them to participate. We informed the interviewees that they would answer questions regarding systems they worked on, and since most projects were closed-source, they also chose the system themselves. This gave them the chance to choose a system they remember well unintentionally or intentionally, or they could prepare themselves in advance.

Furthermore, they have to open their systems in the beginning so that we could ask them about files in their source code folders without us looking at their systems, which could refresh their memory. We asked general questions about the files we chose randomly (e.g. did you contribute to the file) but not being able to look at the content of the files means that we might have chosen files that were not appropriate to ask about to get the results we are looking for. Although unlikely, there is also the risk that they can look at their systems while answering the questions since the interview is conducted online.

External Validity: One of the stated external threats by [1] is the small sample size, which is also a threat to this study, as our sample size is going to be small, and there is a possibility it will be even smaller than the sample size in [1]. The other external threat is stated as assuming knowledge in a general way. Whereas in our study we are focusing more on the data science concepts which will reduce the effectiveness of this threat, it will not completely remove it.

Conclusion Validity: Most of our interview questions are derived from the questionnaire created by [1]. Although they did a well structured systematic literature review to gather the questions, there might be some threats to the conclusion because of the used ways of classification and grouping the questions. Different researchers may categorise the questions in a totally different way, but we decided to follow the exact categorization for the new added questions to be able to compare our final results with [1]. After interviewing different data scientists who are working in different companies, we think that there may not be a specific definition of the tasks which are assigned to data scientists in different companies. This may cause people with the same title (data scientists) to have different responsibilities which eventually leads to different perspectives. With this investigation, the previously mentioned matter, and the threats to validity, it is difficult to conclude confidently what data scientists need to facilitate their program comprehension.

IV. Results

RQ1: What knowledge about their system do data scientists consider important to remember?

To answer our first research question, we asked our participants about what parts of their systems they consider to be important in general and what types of knowledge they consider to be important to remember according to our questionnaire (IK1-IK3, Table I).

— IK1: Which part of your system do you consider important?

We qualitatively analysed the participants' answers on what part of their system they consider to be important (IK1, Table I). We extracted six codes (data science concepts) and then categorised these codes into the three knowledge sections that we had in the questionnaire. Two participants have mentioned the architecture of the system explicitly, and the rest of the nine participants have mentioned parts of the system (the codes we extracted) that we categorised as part of the Architecture, such as model training, pipeline flow which is a number of steps which will take place in a data driven development processes [22], data engineering, and etc. In other words, every participant had included in their answers a part of their system that belongs to the Architecture knowledge section according to our questionnaire. Two participants had mentioned documentation and logging as an important part of their system which we categorised as part of the Meta knowledge section. One participant mentioned code as an important part explicitly, and four people mentioned parts of the system (e.g. model configuration, model evaluation, model serving) that we categorised as part of the Code section. These categorizations may not be accurate but we categorised them according to our questionnaire to stay consistent.

Summary:

- All participants consider parts of the system that belong to the Architecture to be important.
- Nearly half of the participants consider Code to be an important part of their systems.
- Less than half of the participants consider Meta to be important.

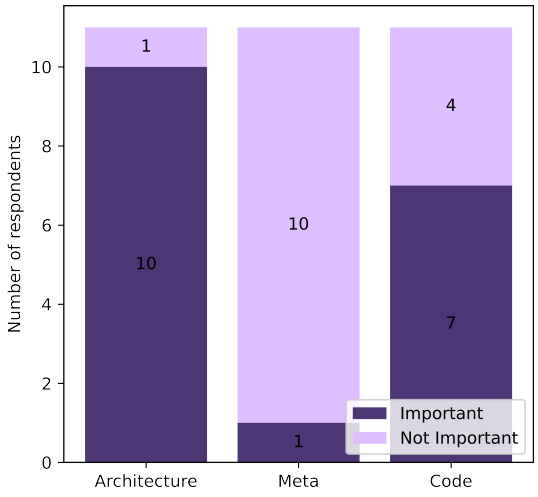
— IK2: Which type of the previously investigated types of knowledge do you consider important?

We asked our participants to choose the knowledge section they think is important (IK2, Table I), reflecting on the questions that we had in the questionnaire. The results in total are: 10 out of the 11 participants chose Architecture as an important knowledge type, and seven out of the 11 participants chose Code, and one out of the 11 participants chose Meta (Fig. 1). Four participants chose Architecture only, and one participant chose Code only. Five of the participants chose Architecture and Code together, and one participant chose all three types; Architecture, Meta and Code.

Summary:

- The majority of the participants believe that parts of the system architecture are important to remember.
- A little more than half of the participants think that Code knowledge is important to remember.
- Less than a quarter of the participants consider Meta to be important to remember.

Fig. 1. Knowledge Importance



— IK3: Which of the previous questions do you consider important or irrelevant when talking about familiarity?

We analysed the answers to this question (IK3, Table I) to understand what the participants consider to be important or irrelevant when they are considering their familiarity with their systems. (All questions we are referring to are in Table I)

Grouping the participants' responses leads to the following results; Architecture questions were mentioned by six of the participants as being important when talking about familiarity. One of them had explicitly chosen learning algorithms (A1), data collection (A3), feature engineering (A5), and hyper-parameter tuning (A6) as important. Another participant talked about general knowledge about model techniques (e.g. A1-A7) to be important to remember. The rest of the participants chose the whole Architecture section as important.

When it comes to the Code knowledge, three participants in total talked about Code being important. One of them explicitly said that "Code" is important for detailed knowledge. The second participant said that code smells (C2) and feature vectors/labels are important (C3). Lastly, the third participant only mentioned that feature vectors/labels (C3) are important. Although code smells (C2) were mentioned as important by one of the participants, another participant referred to the same question but for being not important. On the other hand, six participants chose Meta questions as not important, stating that they questions such as the size of a file (M4) are irrelevant to remember, and it was pointed out explicitly by three participants. Also, one participant mentioned remembering an old file has been rarely or frequently changed (M1) is not important.

Summary:

- More than half of the interviewees chose Architecture questions to be important to remember.
- The majority of the interviewees mentioned Meta questions as not important questions when talking about familiarity.
- The majority of the participants think that the size of a file is irrelevant to remember.
- Feature vectors/labels considered to be important to remember by two participants.

RQ2: Can data scientists correctly answer questions about their system based on their memory?

To assess how well data scientists can remember different parts of their systems, we calculated the averages of the overall correctness of each question, each knowledge section, and each participant.

Architecture: Most participants have scored noticeably high in the Architecture section, as we can see in (Fig. 2) that the average correctness of the architecture section is 92%. The highest average correctness is 100% and it was scored on A2 which asks about the source of the data used in the system. This makes A2 (Table I) one of the most correctly answered questions in the questionnaire. The lowest average correctness is 86% and it was scored on two questions; A5 (Table I) which asks about feature engineering, and A6 which asks about hyper-parameter tuning.

Meta: The Meta section has the lowest correctness, as we see in (Fig. 3) that the average correctness of the Meta section is 64%. The highest average correctness in this section is 100% and it was scored on M5 (Table I) which asks about whether the model in the system had an over-fitting / under-fitting. This makes M5 the second most correctly answered question in the whole questionnaire (the first one is A2). The lowest average correctness is 9% which was scored on M4 (Table I) which asks about the size of a file in terms of lines of code. This makes M4 the most wrongly answered question in the whole questionnaire.

Code: Similar to the Architecture section, the Code section has a considerably high correctness of 92%, with a difference of 0.01% from the Architecture correctness as we see in (Fig. 4). The highest average correctness in this section is 98% and it was scored on C1 (Table I) which asks about the intent of two different files in the system. The lowest average correctness is 84% and it was scored on C2 (Table I) which asks about code smells in the same files that were used in C1.

Participants: The overall average correctness of the 11 participants is 83%. The highest average correctness is 100% and the lowest average correctness is 66%. The rest of the participants have scored an average correctness of 72% to 90%.

Summary:

- Only one participant could remember all the parts we asked about their system correctly.
- Only one participant could remember the size of a file in the system correctly.
- All participants could remember the source of their data, and whether their model had an over-fitting / under-fitting.
- Participants seem to remember the intent of their files, as well as their feature vector and labels better than code details such as code smells and model assessment techniques.
- Participants performed almost equally good in both Architecture and Code with only a difference of 0.01%.

Fig. 2. Architecture Correctness

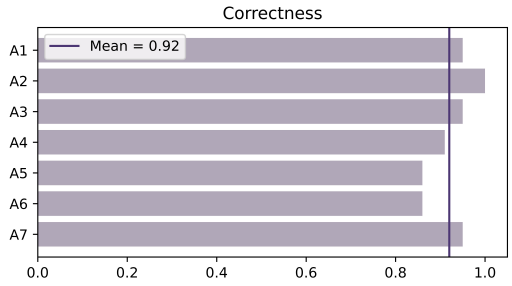


Fig. 3. Meta Correctness

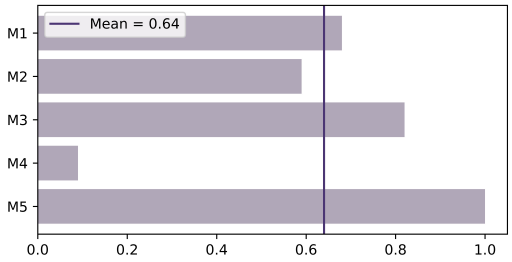
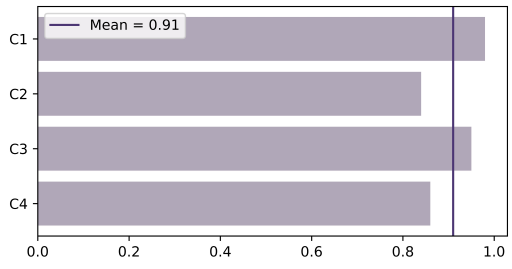


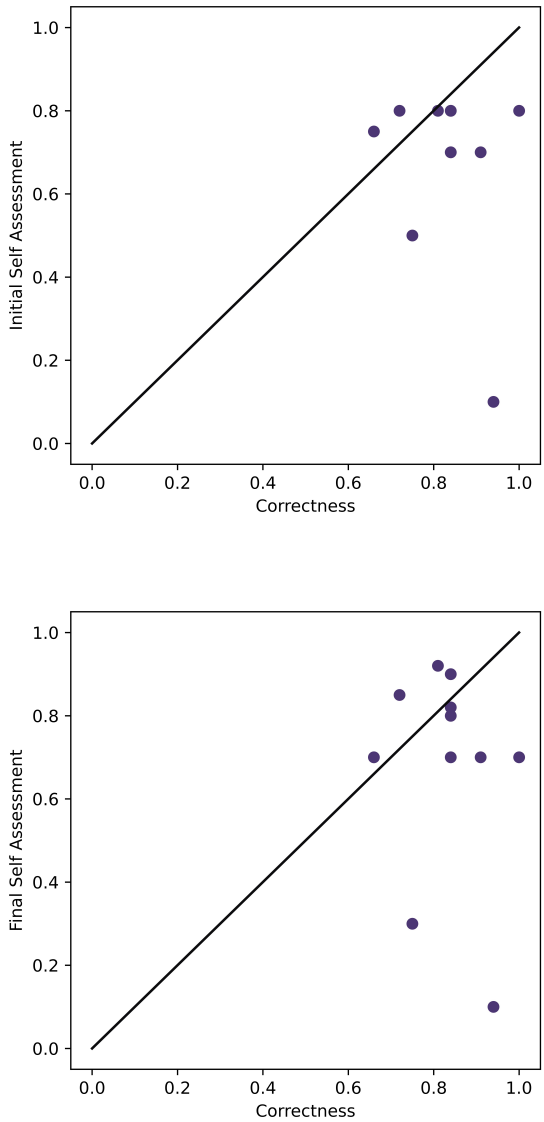
Fig. 4. Code Correctness



RQ3: To what extent does a data scientist's self-assessment align with their actual knowledge about their system?

In order to answer this research question we compared the interviewees' average correctness and their initial and final self assessment results to investigate the presence of any correlation. We also looked into how the results of the initial and final overall self assessments changed during the interview. Analysing the interviewees' initial and final results for overall self assessment shows that five interviewees increased and three of them decreased their overall self assessment. Three interviewees left their self assessment unchanged during the interview. We applied Kendall's τ test on this data based on [1] which results in no significant correlation between the overall self assessment and the average correctness of interviewees. (p-values > 0.05, initial $\tau = 0.242$, final $\tau = 0.061$)

Fig. 5. Self-assessment and Correctness



The participants' responses to IK4 (Table I) were analysed in order to find what they based their overall self-assessment on. Two participants talked about reflecting on the pipeline flow and model training. Two other participants said that

they assessed their ability in explaining and talking about the structure of their systems. Two participants reflected on system structure and the idea behind the implementation of the system. One participant mentioned reflecting on how well they know the system architecture explicitly. We considered all of the points above to fall under the Architecture knowledge section. So in total, we conclude that six participants were considering their knowledge about Architecture when answering the self assessment questions. Three of these six participants also talked about reflecting on Code knowledge. Two other participants talked about reflecting on what they think is relevant in their systems. One participant made their self-assessment based on their confidence in what they remember and do not remember about their systems. Lastly, one of the participants mentioned that they were not considering anything in particular because the system was small and fairly recent, and they were the only developer.

RQ4: What are the similarities and differences between data scientists and developers when it comes to what they consider to be important to remember?

The results in the replicated study suggest that software developers consider Architectural and abstract (e.g. intent) knowledge to be important to remember. The results also suggest that although developers believe Code knowledge is important, it is less important than Architectural knowledge, and developers are not interested in remembering code details such as method parameters. Meta knowledge is considered generally to be less important than Architecture and Code for smaller systems with a few developers [1].

TABLE III

Knowledge Type	Software Engineers	Data Scientists
Architecture	76.5%	90.9%
Code	52.9%	63.6%
Meta	11.8%	9.1%

Results Comparison

We found the results from both studies (Table III) to be almost identical for the following reasons:

- The order of the importance is the same; Architecture followed by Code and lastly, Meta.
- Architectural knowledge is considered to be the most important.
- Abstract knowledge (e.g. system structure, implementation ideas, code intent) is considered important
- Code knowledge such as intent is considered fairly important but not code details
- Meta is considered not important and irrelevant

V. Discussion

A. *Importance of Knowledge*

Based on the analysis of IK1-IK3 (Table I), we understand that:

Architecture is deemed to be the most important part to remember and in general since it was chosen by the participants the most. However, a few participants asked about what we meant by Architecture in the questionnaire. One interesting observation is that one of the participants said that the word "architecture" is not exactly used to represent the set of questions we had in the architecture knowledge section. The participant said that "project settings" would make a better word. In other words, it appears that some terms that are used in computer science and software engineering may not be applicable to data science, especially if the data scientist has no roles that involve other tasks such as developing other parts of the system that are not data science-related. However, this would not be the case for data scientists who are also the software developers of their systems. Thus, research on the differences between architecture in data science and architecture in software engineering should be considered.

Code seems to be considered fairly important based on the results, but we suspect that our categorization of C3 (feature vectors/labels) and C4 (model assessment) was not accurate enough, as we think that there might have been an overlap between architecture and code in questions C3-C4 (Table I). We can also see from the results that when asked about knowledge importance based on the questions we had in the questionnaire (IK3, Table I), only three participants mentioned Code and two of them mentioned C3 explicitly. One interesting remark that one of the participants gave is 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. With all these observations, it is difficult to conclude how important Code is for data scientists. Thus, we think that an extension of our study with better Code categorization is needed.

Meta is considered generally irrelevant and consequently, the least important to remember. Most of the systems that were used in the interviews were small with a number of developers that ranged from one to seven, with the exception of one medium sized system that had 15 developers. We think that the size of the system might have a role in this perception of Meta knowledge, and this result backs up the observation in [1] that research on the importance of meta knowledge in different contexts may be needed.

All in all, we found that abstract knowledge which includes architectural knowledge (e.g. system structure, system behavior) and non-detailed code knowledge (e.g. code intent, implementation ideas) are considered the more important parts of a system to remember, as opposed to detailed knowledge such as code smells. Abstract knowledge concerns the general understanding of data scientists or developers on how and why their systems or certain parts of the systems work the way they do.

B. *Correctness*

The majority of the participants have performed quite well in general, as we see that the overall average correctness of all participants is 83%. Only one participant could remember

all the parts we asked about their system correctly, who also was the only participant able to remember the size of a file correctly. Meta questions were answered fairly well despite being considered irrelevant. We suspect that the question that asks about over-fitting/under-fitting (M5, Table I) which was answered correctly by all participants was not placed in the correct knowledge section and should have been part of the Architecture section. This was also mentioned by one of the participants in the second half of the data collection phase. However, even if we remove M5 from the Meta questions, the average correctness would still be on the higher side. As mentioned before, this could be the result of the majority of the systems being small.

The participants particularly scored high in the knowledge sections which they thought are important; Architecture, followed by Code. Although the results suggest that the participants believe Architecture to be more than Code, they performed almost equally good in both knowledge sections with only a difference of 0.01%. We noticed that the participants seem to remember the intent of their files, as well as their feature vector and labels better than code details such as code smells and model assessment techniques.

C. *Self-assessment & Correctness*

The majority of participants seem to consider their architectural knowledge when assessing their overall knowledge of their systems. Nearly half of the participants increased their overall self-assessment. Less than half of the participants did not change their overall self-assessment. Less than half of the participants decreased their self-assessment. We did not find any specific correlation between the average correctness and the results of overall self-assessment.

D. *Threats to Validity*

We think that the Architecture-Code overlap we mentioned previously (refer to Discussion: A) might have had an effect that led to this equally good performance in both Architecture and Code. We also think that the high average correctness of Architecture questions might be a result of a few participants who did not check their repository during self-correction properly, and their self-correction was mostly based on confidence in what they remember. Not being involved in evaluating the correctness of the answers might also have had a role in the results. We think that it is difficult to come to a conclusion confidently based on these results. Therefore, doing the exact same research but with open source systems which the interviewers can access should be considered.

E. *Results compared to the replicated paper*

Based on the results of the original study and our study:

Knowledge Importance: Both data scientists and software developers think that Architectural knowledge to be the most important to remember, followed by Code. Both data scientists and software developers (of smaller systems) seem to agree on Meta knowledge not being important to remember.

Correctness: The lowest average correctness of a question is scored in the question that asked about the file size, which is similar to the result in the original paper, indicating that both data scientists and software developers do not place importance on remembering some Meta knowledge, which is also supported by their remarks on the irrelevance of Meta when talking about knowledge importance. Another similar result is that when it comes to Code knowledge, both data scientists and software developers performed best on questions about the code intent of their system files. In addition, the average correctness of the knowledge sections is the same in both studies; with Architecture having the highest average correctness and Meta having the lowest.

Self-assessment: Interestingly in our study, the number of participants who increased their overall self-assessment is less than the ones who decreased it, which is contrary to the results from the replicated study where the number of interviewed software developers who decreased their overall self-assessment were more than those who increased it. We think that this difference in the results supports [1]'s statement on the need for research that investigates the reliability of self-assessments in more detail and the consideration of guidelines for research methods.

F. *Additional Remarks*

We received some of the interesting insights from the additional remarks. One participant mentioned that our questionnaire lacked data-quality questions, and another participant argued that data knowledge is more important than Code specific knowledge or Meta knowledge. Two participants mentioned that "code smells" are not as important to them as they are to software developers. Although this could be a subjective point of view and not necessarily related to being data scientists, we thought that their remark was interesting as taking code smells into consideration is a well-known good practice for software developers.

By looking at these observations and the backgrounds of the participants, we believe that there might be differences between the viewpoints of the data scientists who are working in the industry and those who are working in academia. We think that an extension of this research that includes data knowledge questions in the questionnaire with a more accurate knowledge section categorization is needed to better clarify data scientists' information needs so that support tools and techniques for data scientists can be appropriately addressed and looked into.

VI. Conclusion

To conclude, we studied the types of knowledge data scientists consider to be important to remember, if they can correctly remember parts of their system from memory and whether there is a correlation between their correctness and the type of knowledge they consider to be important. We also studied how well their self-assessment aligns with their actual knowledge and observed the changes of their self-assessments during the interview. Based on the data we received from the

11 data scientists who are working on small systems (with the exception of one medium sized system), we found that:

- **Data scientists believe that Architecture is the most important type of knowledge to remember, followed by Code. They do not place importance on Meta knowledge. Similar to the software developers based on the results from the replicated paper.**
- **Data scientists increase their self-assessment, which is contrary to Software developers based on the results from the replicated paper.**
- **Data scientists answer correctly best in the questions that they believe are more important.**

In addition to all these points, we want to extend this study and involve data scientists with different backgrounds and from different companies to gain more insight on their responsibilities. We also would like to interview data scientists who are working on larger, or open source systems. Furthermore, after learning more about data science and the potential miscategorizations we had, we want to improve the questionnaire for the future extended studies.

VII. Acknowledgments

We thank our supervisor Regina Hebig for the great supervision and support we received. We also thank all of our participants for their time and the interesting insights they provided.

References

[1] J. Krüger and R. Hebig, "What Developers (Care to) Recall: An Interview Survey on Smaller Systems," in *2020 IEEE International Conference on Software Maintenance and Evolution*, 2020, ser. ICSME, pp. 46-57, doi: 10.1109/ICSME46990.2020.00015.

[2] L. McLeod and S.G. MacDonell, "Factors that affect software systems development project outcomes: a survey of research," in *2011 ACM Computing Surveys*, 2011, vol. 43, no. 4, pp.24-56, doi: 10.1145/1978802.1978803.

[3] W. Maalej, R. Tiarks, T. Roehm, and R. Koschke, "On the Comprehension of Program Comprehension," in *2014 ACM Trans. Softw. Eng. Methodol*, 2014, vol. 23, no. 4, 37 pages, doi: 10.1145/2622669.

[4] D. Graziotin, F. Fagerholm, X. Wang and P. Abrahamsson, "What happens when software developers are (un)happy," *Journal of Systems and Software*, 2018, Vol 140, pp. 32-47, ISSN 0164-1212, doi: 10.1016/j.jss.2018.02.041.

[5] M. Lavallée and P. N. Robillard, "Why Good Developers Write Bad Code: An Observational Case Study of the Impacts of Organisational Factors on Software Quality," in *IEEE/ACM 37th IEEE International Conference on Software Engineering*, 2015, ser. ICSE, pp. 677-687, doi: 10.1109/ICSE.2015.83.

[6] B. Fitzgerald, "An Empirically-Grounded Framework for the Information Systems Development Process", 1998, ser. ICIS, [Online]. Available:https://aisel.aisnet.org/icis1998/10, Accessed: 2022.

[7] J. Winkler and A. Vogelsang, "What Does My Classifier Learn?" A Visual Approach to Understanding Natural Language Text Classifiers, 2017, doi: 10.1007/978-3-319-59569-6-55.

[8] H. Liu, S. Eksmo, J. Risberg, and R. Hebig,"Emerging and Changing Tasks in the Development Process for Machine Learning Systems," in *International Conference on Software and System Processes*, 2020, ser. ICSSP, Association for Computing Machinery, New York, NY, USA, pp. 125–134, doi: 10.1145/3379177.3388905.

[9] S. Amershi et al., "Software Engineering for Machine Learning: A Case Study," in *IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice*, 2019, ser. ICSME-SEIP, pp. 291-300, doi: 10.1109/ICSE-SEIP.2019.00042.

[10] J. Linåker, S. Sulaman, M. H. Ost and R. M. de Mello, "Guidelines for Conducting Surveys in Software Engineering," 2015.

[11] M. Kasunic, "Designing an effective survey," 2005.

[12] A. Burkov, The Hundred-Page Machine Learning Book, [Online]. Available: http://themlbook.com/wiki/doku.php, Accessed: 2022.

[13] S. Haiduc, J. Aponte and A. Marcus, "Supporting program comprehension with source code summarization," in *ACM/IEEE 32nd International Conference on Software Engineering*, 2010, pp. 223-226, doi: 10.1145/1810295.1810335.

[14] R. Montavon, W. Samek, K. Müller, "Methods for interpreting and understanding deep neural networks," in *Digital Signal Processing*, 2018, pp. 1-15, doi: 10.1016/j.dsp.2017.10.011.

[15] T. Roehm, R. Tiarks, R. Koschke and W. Maalej, "How do professional developers comprehend software?," in *34th International Conference on Software Engineering*, 2012, ser. ICSE, pp. 255-265, doi: 10.1109/ICSE.2012.6227188.

[16] J. I. Maletic and A. Marcus, "Supporting program comprehension using semantic and structural information," in *23rd International Conference on Software Engineering*, 2001, ser. ICSE, pp. 103-112, doi: 10.1109/ICSE.2001.919085.

[17] X. Xin, L. BAO, D. Lo, Z. Xing, A. E. Hassan, S. Li, "Measuring program comprehension: A large-scale field study with professionals," in *IEEE Transactions on Software Engineering*, 2018, vol. 44, no. 19, pp. 951-976. Research Collection School Of Information Systems.

[18] M. Kim, T. Zimmermann, R. DeLine, A. Begel, "The emerging role of data scientists on software development teams," in *38th International Conference on Software Engineering*, 2016, ser. ICSE, pp. 96–107, doi: 10.1145/2884781.2884783.

[19] P. Pereira, J. Cunha and J. P. Fernandes, "On Understanding Data Scientists," in *2020 IEEE Symposium on Visual Languages and Human-Centric Computing*, 2020, ser. VL/HCC, pp. 1-5, doi: 10.1109/VL/HCC50065.2020.9127269.

[20] P. Pereira, "Towards Helping Data Scientists," in *2020 IEEE Symposium on Visual Languages and Human-Centric Computing*, 2020, ser. VL/HCC, pp. 1-2, doi: 10.1109/VL/HCC50065.2020.9127198.

[21] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, "Understanding Neural Networks Through Deep Visualization", 2015, doi: 10.48550/arXiv.1506.06579.

[22] A. De Lucia, E. XheloR, R, Smedinga, M. Biehl, "Data Science Pipeline Containerization", 17th SC@RUG 2020 proceedings 2019-2020, 2020, Bibliotheek der R.U.