Human-Centered Design for Microservice Recommendation

MELANIE BANCILHON[*], Washington University in St. Louis, USA
RAHUL KRISHNA, IBM Research, USA
JOHN ROFRANO, IBM Research, USA

An essential component of application modernization is the decomposition of monolith architectures to microservices, which has become a standard for designing enterprise applications. Various recommendation tools have been developed to facilitate this task but they may produce inaccuracies or suggestions that misalign with developers' intentions. While human-centered design has been adopted across various research areas to examine interactions between human and AI and create tools that harness their respective strengths, its adoption in the field of microservice recommendation has been largely overlooked. In this work, we examine the current challenges faced by developers when using microservice recommendation tools and propose UI features that strive to improve their interactions. We conducted interviews with 8 software engineers that sparked discussions about information overload, the importance of communicating uncertainty and opportunities for human-AI collaboration. Based on our findings, we developed and evaluated an interactive mixed initiative interface that supports the design of microservices. We report our observations and discuss the importance of human-centered research in application modernization.

CCS Concepts: • **Human-centered computing** → **HCI theory, concepts and models**; • **Software and its engineering** → *Designing software*; • **Computing methodologies** → Generative and developmental approaches.

Additional Key Words and Phrases: microservice recommendation, human-AI collaboration, visual interactive interfaces, application modernization

ACM Reference Format:
Melanie Bancilhon, Rahul Krishna, and John Rofrano. 2018. Human-Centered Design for Microservice Recommendation. In . ACM, New York, NY, USA, 14 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

One crucial aspect of application modernization is the decomposition of monolith architectures to microservices, a task which enables businesses to better manage, deploy, and scale their applications. Since manual approaches are time-consuming and expensive, using a tool that provides partitioning recommendations is often a more feasible approach for developers. While these tools have become increasingly accurate due to the rapid advancements in AI, they almost always operate at less than perfect accuracy and can produce undesirable outcomes such as distributed transactions, or recommendations which may deviate from developers' intentions. Studies have shown that the vast majority of users make changes to partitions recommended by microservice tools [11, 21].

Across a number of applications, human-AI collaborative approaches have been used to refine imperfect systems. Prior work has demonstrated their ability to enhance speed, accuracy, and decision-making across various domains. Numerous mixed-initiative tools have been developed across diverse applications, ranging from medical decision

[*]This research was conducted while this author was an intern at IBM research

support to data labeling. Within the realm of visual analytics, mixed-initiative visualizations have played a crucial role in enabling seamless and intuitive interactions with algorithms, thereby facilitating more effective reasoning. A number of studies have shown that the design choices of interfaces and tools utilized to support human-AI interactions significantly impacts usability, comprehension, and overall performance [22].

Despite prior work demonstrating the benefits of human-AI collaboration, their adoption in the realm of microservice recommendation still remains unexplored. Our research is focused on gaining insights into the current challenges faced by software developers when using microservice recommendation tools. Furthermore, we aim to identify user interface (UI) features that can enhance human-AI interaction and inform the design of a mixed-initiative tool for microservice recommendation.

We conducted formative interviews with 8 industry professionals and through a thematic analysis, and identified three common themes: information overload, granularity and explainability. Our interviewees expressed the need of seeing a more granular view of their application while highlighting a concern for information overload, which can result in an overcrowded display, making it confusing and difficult to navigate. Although they deemed useful to view the algorithm's confidence, they found it unnecessary to receive explanations about the algorithm's mechanisms. We propose a human-in-the-loop framework that augments existing AI-based algorithm CARGO [17] and serves as a usage scenario to frame our design and evaluation. Then, we design an interactive mixed-initiative interface with UI components that strive to address the pain points raised by users in the formative interviews. We evaluated our interface via semi-structured interviews with the same 8 participants, where we asked them to complete a set of tasks using a think-aloud protocol to observe how they understand, use and interact with our various UI components. Users expressed enthusiasm for the functionality of the various features implemented, and were able to successfully explore classes based on the uncertainty of their partition assignment, understand how the uncertainty value was assigned to classes, use filters and search functions, alternate between class and method displays and split methods across classes. However, participants failed to correctly interpret in-the-loop partition recommendations, and failed to correctly detect methods in our graphical display. We discuss the implications of these findings in relation to our design choices and argue for the development of collaborative microservice recommendation algorithms.

The contributions of our research includes the following:

- We gain understanding into practitioner workflow and identify challenges when using microservice recommendation tools.
- We propose a human-in-the-loop framework to augment existing AI-based algorithm CARGO to capture user interactions.
- We develop and evaluate an interface that improves human-AI interaction and supports their collaboration for microservice recommendation.
- We discuss the implications of our approach and argue for more human-centered research in application modernization.

2 RELATED WORK

Our work is situated within HCI research that strives to understand user needs to inform the design of AI-powered tools. In this section, we review and discuss two areas relevant to our work: the application of human-AI collaboration to microservice recommendation and the design of mixed-initiative interfaces.

2.1 Human-AI Collaboration for Microservice Recommendation

Microservice recommendation tools are often a more feasible approach to decomposing a monolith to microservices compared to manual approaches, which are time-consuming and expensive. There exists a number of patterns for decomposing an application, one of the most prominent being identifying functional boundaries in the application that are also loosely coupled to other functionalities in the code. Several applications AI-guided partitioning to implement this pattern [1, 9, 11]. For example, Mono2Micro [11] performs static analysis of the application to obtain structural information and data dependencies, which are then analyzed by an AI engine to create partitioning of application classes that considers business use case similarity and call dependencies. Another algorithm, which also takes into account database dependencies, is CARGO, short for Context Sensitive Label Propagation [17]. CARGO builds upon the principles of Label Propagation Algorithm (LPA) to build a System Dependency Graph (SDG) used to create highly accurate partitions [17]. It has been shown to mitigate distributed transactions, improve run-time performance and performance on architectural metrics compared to other algorithms that only consider functional boundaries. Several other microservice recommendation techniques are reviewed in a survey by Ponce et al [19].

Although they are more efficient, microservice recommendation algorithms can sometimes produce inaccuracies such as distributed transactions, or partitions that do not align with expert's intentions. Kalia et al. interviewed 20 professionals about the usefulness of Mono2Micro and found that throughout their experience using Mono2Micro, only 7% of users did not make any changes to the algorithm's recommendations, while 30.8% made a few minor changes, 38.5% made some minor and some major changes and 23.1% made many major changes [11]. The most prevalent changes to Mono2Micro's recommendations included (in order of most to least prevalent) moving classes, adding new classes, creating new partitions and renaming partitions. In order to leverage the respective strengths of human and AI, various domains have proposed human-AI collaborative systems. Several studies have shown that human-AI collaboration can improve speed [2], accuracy [2] and decision-making [3]. For example, Ashktorab et al. found that AI-assistance in data labeling tasks, in which a human annotator makes decisions for which labels to apply to data, sped up the data labeling process and also increased the accuracy of data labelers [2]. To the best of our knowledge, this work is the first to propose a human-AI framework for microservice recommendation.

2.2 Designing Mixed-Initiative Interfaces

Designing interfaces and tools that support human-AI collaboration is not trivial. Literature across disciplines has demonstrated that both the nature and format of the information communicated can impact how users perceive and interact with various systems. In the area of explainable AI, several studies have shown that different types AI explanations may naturally show distinctive impact on human decision makers [Wang et al]. Researchers in the field of visual analytics have developed various tools that strive to facilitate interactions between human and machines for a number of tasks [4–8, 10, 12, 14, 18]. Monadjemi et al. found that when interacting with a visualization in a guided data discovery task, participants tended to ignore recommendations despite their relevance to the task. The authors argued that the presentation style of the recommendation might have caused this effect and highlight the importance of investigating different ways of presenting suggestions [15]. Whitworth et al. argues that making suggestions obtrusive can cause users to ignore or disable them, a known example being prior assistance agent Clippy built by Microsoft [24]. While investigations into the design of human-AI collaborative tools span various applications, less attention has been given to the area of application modernization. Some studies have focused on code translation [20, 22, 23]. For example, Weisz et al. interviewed a group of software engineers to examine the effect of UI features on the adoption of

AI-generated code translation [22]. They found that overall, participants most valued the designs where translation confidence was communicated and asserted that the latter was critical in prioritizing review effort [22].

When it comes to migrating applications from monolith to microservices, several tools use graphical user interfaces and visualizations to assist users in this task [11**?**]. For example, Nakazawa et al. has developed a tool to design microservices with the monolith-first approach [16] that uses a node-link diagram. Another example is enterprise tool Mono2Micro [11] that uses interactive visualizations. Despite their use of graphical user interfaces and visualizations, their design choices are most often arbitrary. To the best of our knowledge, there exists no studies on the design of human-AI collaboration for microservice recommendation.

3 FORMATIVE END USER INTERVIEWS

In order to better understand current workflow and pain points with current microservice recommendation tools, we conducted formative interviews with knowledge workers responsible for microservice development or managing teams that develop microservices at an enterprise level. The questions spanned the following themes: expert role and tasks, experience with refactoring, experience with tool features and interactions, opinions on an ideal microservice tool. We expanded on the level of detail they desired when partitioning code, the UI interactions and features. Two of the authors conducted a thematic analysis, consisting of identifying, annotating and extracting common responses and themes from the interviews. They reviewed meeting recordings and transcripts and identified high-level themes, and refined them through collaborative discussions. They summarize their main insights in this section.

We recruited 8 full-time software engineers experts within our organization that have at least some experience with microservice recommendation tools. Participants represented a diverse range of roles across various organizations which are listed in table 1.

ID	Role	Organization	Manager?
P1	Software Engineer	Software	N
P2	Software Engineer	Software	N
P3	Lead Modernization Architect	Finance & Operations	N
P4	Lead Modernization Architect	Finance & Operations	N
P5	Integration Technical Specialist	Sales	N
P6	Application Architect	Consulting	N
P7	Senior Technical Staff Member	Finance & Operations	Y
P8	Senior Technical Staff Member	Global Sales	N

Table 1. The table records our interviewees' ID, role, organization name and whether they are a manager or not.

3.1 Information Overload

A prominent topic that was brought up by participants is the issue of information overload, especially for large applications. P6 mentioned that their application *"had a lot of classes and it was hard to figure out what was going on"*. P3 mentioned that the size of the application is an important contributor to the information overload problem *"If you have a three microservice application, it's consumable, but when you have like a fifty or, you know, sixty it starts to get really confusing. So having more summary pages, you know, have things in red like problem here, green here, you know"*.

They also brought up the usefulness of filtering several components of the view, such as the edge types and edge weights. P1 and P2 mentioned the usefulness of the table view and they can more easily find a class compared to the graph view, where they have to hover over each node. P2 stated that *"In the table view, I can search for a class more easily but in the graph view, I have to hover over every node to find a class"*.

The topic of information overload has been extensively studied in the visualization community [cite]. One of the most influential mantra by Schneiderman is "overview first, zoom and filter, details on demand".

3.2 Customization and granularity

We asked users questions about the level of granularity at which they would prefer to see their application. One theme that was brought up was the importance of database interactions, which is at the center of the CARGO algorithm. P4 stated that *"you can have a really good microservice design but it can all fall apart if you don't design how you interact with the database correctly"*.

In their current application, users can see their applications in the form of classes. When addressing our question about the desired level of granularity, responses varied. However, it is important to highlight the interplay between desired level of granularity and the information overload issue. Several participants P2 mentioned that *"for my application, classes are granular enough. Showing methods would be too messy"*. They highlighted that the user perhaps could opt into seeing method as long as they don't overcrowd the display. P3 mentioned that *"Seeing methods would be useful as long as it's kind of like look here in this class, there's some problems in here and then you can open it up. See where the problems are like, it allows you to drill down, but let people opt into that complexity"*. We posit that the desired level of granularity depends on several factors including the size of the application, level of familiarity with the application and experience of the developer. Prior work highlights the potential benefit of creating partitions at the function level [16].

3.3 Considerations for Explainability

This line of questioning was highly influenced by Weisz et al. [22], who inquired about developers acceptance and utility of imperfect AI. While different microservice recommendation tools employ different clustering strategies, most of them including their current industry tool are deterministic. Moreover, the current microservice tool does not provide the user with meaningful partition names based on its strategy. Therefore, we asked participants to comment on the importance and utility three aspects of microservice recommendation: (1) communicating uncertainty, (2) meaningful partition names and description and (3) comprehension of AI mechanics.

In general, participants were enthusiastic about a system that shows classification confidence. As stated by P1 "showing uncertainty for classes would be useful cause you can just ignore some and focus on the borderline cases". Similarly, P4 mentioned the importance of indicating where there is a problem in the partitioning. P2 mentioned that *"If you could say you are partitioning this together because they are part of this functionality, and then give me a short explanation of what it is, I think that would be very useful"*. When it comes to the importance of assigning meaningful partition names and definition, once again we found that responses varied based on the size and the developer's familiarity with the application. While P1 mentioned that he is so familiar with his application that he can just glance at a partition and tell which business use case it represents, other users stated that it would be a very useful feature. P2 mentioned *"It would be very useful to see partition name and description. It would help me understand why they put some classes in this partition."* Users considered the comprehension of AI mechanics to be unnecessary and potentially confusing.

4 USAGE APPLICATION: HUMAN-IN-THE-LOOP WITH CARGO

As mentioned in 2.1, an array of algorithms have been developed to recommend functional partitions in monolithic code. We posit that these algorithms would benefit from leveraging the expertise that developers have about their applications to improve the accuracy of their recommendations. In this section, we demonstrate how current algorithms can be augmented into collaborative tools by proposing a framework for CARGO, an algorithm recently developed by Nitin et al. [17] which leverages AI techniques to improve the partitioning quality of current algorithms.

Overview of CARGO. CARGO creates microservices using three steps (1) Initiation with seed labels, (2) Initial Label Propagation Algorithm (LPA) and (3) Iterative LPA over context snapshots. CARGO provides the flexibility of either being fully unsupervised and assign seed labels randomly, or semi-supervised, by providing seed labels from an existing partition assignment. A LPA is then applied to the program dependency graph's transactional snapshots and seed labels are factored in. The nodes of the program dependency graph represent the methods and database tables in the application. The LPA is then iteratively applied to context snapshots to obtain partition assignments for methods. Method assignments are then aggregrated to obtain class assignments. While in-depth knowledge is not required for the purpose of this framework, more information can be found in the paper [17].

Human-in-the-loop Scenario. When using a microservice tool, a software developer first starts the process of decomposing their monolith by exploring the partitions recommended by the underlying algorithm. While level of granularity at which portions of the code base are partitioned vary across tools, most existing tools operate at the class level for easier manageability. CARGO assigns partitions to methods, which are then aggregated into class partitions. A user is likely to identify some misassigned classes based on their knowledge of the application. They start editing the recommendation by moving a class from one partition to another. In our human-in-the-loop framework, we propose to capture this interaction and pass it as a seed label in step (1). CARGO will then generate a new set of recommendations based on the user's expertise of the application. If the generated assignments differ from the original assignments, new assignments will get suggested to the user, who can choose to approve or reject the recommendation, thus creating a closed feedback loop between the user and the algorithm. Figure 1 illustrates our proposed framework for CARGO.

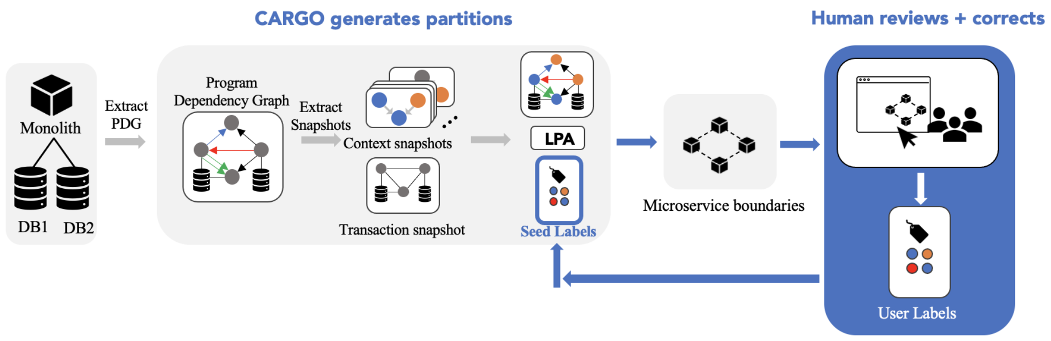

Fig. 1. Our proposed human-in-the-loop framework for CARGO [17]

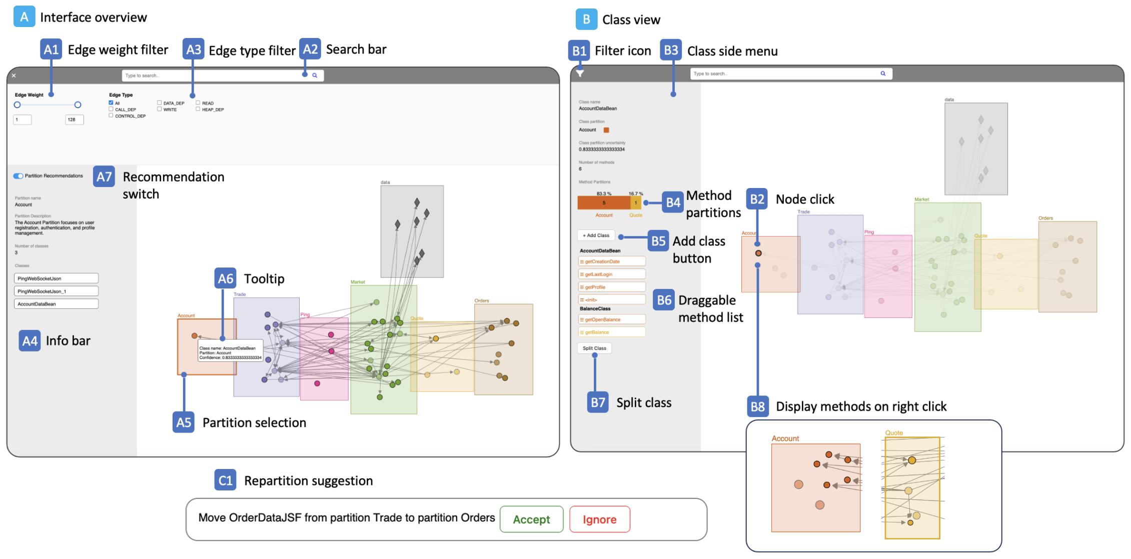

Fig. 2. An overview of CARGO's interface. A By selecting the filter icon B1, the filter menu pops up at the top of the page. The user can filter the volume of calls between nodes using the edge weight filter A1 and the nature of calls between nodes using the edge type filter A3. The user can search for a class or data table using the search bar A2, which will select the relevant node as shown in B. In A, the interface shows a selected partition A5 and its properties in the info bar A4, which includes its name, description, number of classes and class names. The user can hover over a node to display a tooltip with its name, partition and confidence A6. B shows the interface when a class is selected B1. Properties of the selected class appear in the info bar, including a bar chart indicating its methods assigned partition B4 and color coded draggable list of its methods B6. The user can choose to split the class by using the add class button B5, dragging methods from the method list B6 into the new class and clicking on the split class button B7. Users can also have a graphical overview of the methods by right clicking on a class node. Methods will appear as smaller circles within their corresponding partitions B8. If the recommendation toggle is on A7, the user can receive suggestions as they start moving classes from one partition to another C1.

5 DESIGN CONSIDERATIONS & USER INTERFACE

Based on our findings from the formative interviews, we developed a graphical user interface with features that strive to address user pain points and needs reported in Section 3. To provide better contextual understanding and evaluation, we anchor our user interface on our human-in-the-loop framework for CARGO, described in 4. While our interface does not contain all the functionalities of a comprehensive microservice recommendation tool might have, it serves as a prototype to identify how users interact with the desired features.

5.1 Overview

Walkthrough: Priya starts using our interface to refactor her application to microservices.

As shown in figure 2, classes, databases and their partitions are represented through a node-link diagram, where rectangles represent partitions, circles represent classes and diamonds represent tables. Classes are contained within different partitions, whose labels can be seen on the screen, and datatables are contained with the "data" partition. Connections between classes, datatables, and classes and datatables can be seen are represented through links. The user can hover over a node to view its name, partition and assignment confidence. We chose to use a node-link diagram to represent partitions to keep the design consistent with current microservice recommendations tools to match users current mental model and minimize their learning curve.

On the left side, there is an info bar A4 where the users can see more information about different elements in the display. The interface overview A, shows the display after the user selected the *Account* partition. The info bar A4 shows a toggle button where users can turn human-in-the-loop recommendations on or off, a description of the partition, derived from open source LLMs, namely GPT-4 and StarCoder, as well as a list of all the classes within that partition.

5.2 Supporting drill-down exploration.

Walkthrough: Priya is interested in exploring the calls between classes, and between classes and datatables. More specifically, she is interested in knowing which classes interact with each other the most, and which classes write to database tables the most. Priya is also interested in identifying the Log class in her application.

In order to mitigate information overload, we enable user to filter out edges on the graph by the nature of calls, and the volume of calls between classes and data. We also allow users to search for a particular class or data table.

There exists several techniques to visualize edge weight, a common one being by using edge thickness. We chose to maintain a consistent edge thickness to prevent a cumbersome design and used an edge weight filter A1 inspired by [13] where he user can use sliders to select a range for the volume of calls. The node-link diagram will only show edges with the specified parameters and filter out the rest, allowing users to explore their desired call volume without overcrowding the display. The user can filter out the type of calls between classes and datatables by selecting them through the edge type checkboxes A3. The display when filters out the other edges accordingly. To identify a specific class, the user can use the search bar A2 and type in the class name. The search bar will provide suggested nodes as the user is typing.

5.3 Communicating Class Uncertainty

Walkthrough: Priya is interested in knowing how confident the algorithm is in the recommendations provided in the initial display.

Several studies have demonstrated the importance of explainability in human-AI collaboration. In our formative interviews, users expressed the importance of communicating AI outcome uncertainty. In our usage application, there are two levels of uncertainty, namely the uncertainty of CARGO when assigning partitions to methods and the error rate in the class partitions created by taking the mode of the method partitions. Since CARGO is deterministic, further research needs to be conducted on the interpretability of its uncertainty. Therefore, we chose to examine the uncertainty class partition assignments.

We communicate uncertainty for each class through node opacity. The higher the opacity, the more confident the algorithm is in the partition assignment. The user is then able to focus on the least confident node in their reviewing process. Class uncertainty is also visible in the tooltip A6 as users hover over nodes.

5.4 Displaying Method Partitions & Splitting a Class

Walkthrough: Priya identifies a class with low confidence and would like to know how the methods are distributed across partitions.

As described in 4, CARGO assigns partitions based on methods, which are then aggregated to classes. Users expressed caution about displaying methods in a network graph due to the possibility of information overload, which could impact the interface's navigability. However, they also acknowledged the potential advantages of viewing the partition assignment for methods. This would not only enable users to have a more detailed view of their application, but it would also enable them to better understand how class partitions are assigned.

As shown in B, the user can click on a class node B1 to display information about its methods in the info bar. B4 shows a bar chart representing the number and percentage of methods in each partition. The user can also see the method names in a draggable list B6, color coded by their assigned partition. The user can also see a class' methods graphically on the node-link diagram by expanding a class node through a right click interaction B8. The corresponding methods will appear on the display, represented as smaller circle. To collapse the nodes back into the class node, the user can right click on any of its methods.

5.5 Splitting a Class

Walkthrough: After identifying classes with low confidence, Priya needs to decide whether the class needs to be split into two classes.

As previously discussed, some users believed that showing class uncertainty as well as enabling method-level granularity would be useful. Therefore, we deemed useful to enable the user to move methods to a new class if it contains a set of methods from various partitions, causing cross-partition calls. In reality, moving methods from one class to another would require modifying the code base. While our interface does not provide this feature, we investigate how users would go about editing the high level structure of methods and classes.

If the methods are split between two classes and the number of cross-partition calls is too high, the user can decide to split the class into two or more classes. The user can click on the "add class" button B5 to add a new class, rename the classes and then assign methods to classes by dragging and dropping them into the corresponding classes B6. Once users are satisfied with their assignments, they can click on the split class button B7

5.6 Suggesting Partition Recommendations

Walkthrough: Priya wants to move some classes from one partition to another because it better suits the structure of her application. She starts by moving a first class from one partition to another using a drag and drop interaction.

In order to support the user in their editing process, algorithms should be able to learn from user interactions and recommend similar data points. Section 4 describes how such a framework can be implemented with CARGO. When the user moves one class from one partition to another, the interface should capture this interaction and pass it to the algorithm to update its partition assignment recommendation. If the assignment of one or more data points changes, the interface should recommend the user to also move these classes.

After the user moves one class from one partition to another using a drag and drop interaction, a suggestion box appears on the screen C1 if the algorithm generates any changes to class assignments after considering the user's interactions. The suggestion asks the user whether they would like to move the changed data point from one specified partition to another. The user can choose to accept or ignore the algorithm's suggestion. Prior work in labelling and active search have used visual interfaces that guide users in their tasks by recommending relevant data points to investigate and querying the users on whether they would like to accept the system's suggestions. Similarly, in our work, suggestions are displayed at the top of the screen after the user moves one class from a partition to another.

6 PROTOTYPE EVALUATION

Motivated by our design considerations, we evaluated our prototype with the same eight software engineers who completed the formative interviews. We conducted 30-minute guided user studies studies via videocall that we recorded.

In our study, users were asked to complete a set of pre-defined tasks while using a think-aloud protocol. When conducting these tasks, we focus on evaluating the features described in our design considerations based on how participants understand, use and interact with them.

6.1 Study Design

Our scenario used the DayTrader application, a popular Java Enterprise Edition (JEE) trading application [1], with partition assignments generated by CARGO [17]. First, we briefly instructed users on the think-aloud protocol and gave an overview of the task. Then, we shared a link to our instruction page and asked them to share their screens. The instruction page explained each feature in our interface and their functionality. They were then shown the following lists of tasks.

(1) View the definition of the *Account* partition
(2) Identify classes that read and write to database tables
(3) Identify classes that have a high volume of calls to other classes
(4) Find the *MarketSummaryDataBean* class
(5) Identify classes that are likely to belong to another partition
(6) Move *TradeConfig* to the *Orders* partition
(7) Accept the suggestion to move the other node
(8) Expand *AccountDataBean* to see its methods
(9) Collapse the methods back into their class
(10) Split *AccountDataBean* into two classes

Tasks (1), (2), (3) and (4) evaluate the use of the edge filters and class search features. We observe whether user interactions appear to be intuitive and seamless. In task (5), we observe whether users make use of the uncertainty information available to them through the node opacity. In task (6) and (7), we test how the users understand and interact with the partition suggestions the usability of our approach for splitting a class into two classes. In tasks (8) and (9), we observe how the user expands and collapse a class and whether they are able to differentiate methods from classes. In (10), we observe how users split a class into two classes. After reading the instructions and tasks, participants clicked on a button to launch the interface in a new tab. Participants conducted the tasks sequentially, asking for guidance when needed while we took notes of our observations.

6.2 Results

We analyzed the interview transcripts in relation to the set of tasks outlined above and our design requirements. We analyzed the think-aloud transcripts and screen recordings together to identify how users interacted with the interface to complete the set of assigned tasks.

Supporting Drill-Down Exploration. When participants were asked to identify the classes that read and write to database tables (2) and classes that have a high volume of calls to other classes (3), all the participants first attempted to identify the classes by glancing at the node-link diagram. While some successfully completed task (2) using this technique, others raised difficulties in identifying a class due to the overlapping links. After we reminded participants of the filter menu, they were able to use the edge weight filter and edge type filter although three of them highlighted that the filter names are not very informative (P4, P7, P8). P8 stated *"you should rename them to volume of calls and*

[1] https://github.com/WASdev/sample.daytrader7

nature of calls". All participants were able to use the search bar to complete task (4). P6 stated “I like that I can search for the class name and it will show it to me".

Communicating Class Uncertainty. When completing task (5), all participants demonstrated an understanding of class uncertainty. P3 stated before completing this task *"So this has to do with the certainty. If something wasn’t very confident where it should go then maybe it’s a candidate to move somewhere else"*. All participants also successfully interpreted the opacity of the nodes as class uncertainty and glanced at the display to identify the nodes with the lowest opacity while hovering on each node to view the uncertainty value. P1 stated *"So these would have the lighter gradient. . . that makes sense I just have to look at different colors"*. However, P2, P3 and P8 highlighted that an uncertainty filter would be beneficial to identify the nodes with specific uncertainty values in a more straightforward way. P3 stated that they wished to have seen a filter *"Like you had the slider before maybe show me the things you are least or most confident in"*. Participants expressed that the current technique of detecting low uncertainty nodes might be time consuming and tedious for large applications.

Displaying Method Partitions and Splitting a Class. When identifying a class with low confidence (5), users clicked on it to view its methods in the info bar and demonstrated an understanding of the provenance of the uncertainty value. P3 mentioned the potential usefulness of visually seeing the connections between the methods and the other classes, confirming the benefit of visually displaying methods in the node-link diagram, a feature that we provide B8. While most participants successfully expanded a class into methods through a right click interaction (8) and highlighted the benefit of this functionality, when asked to identify the methods in the graph, 7 participants missed one method, which belonged to a different partition than the class. This was due to the lack of visual difference between the representation of classes and methods. P3 suggested using a different shape for methods, like a square, P7 suggested nesting the methods inside the expanded class and P1 suggested using an animation that guides the user’s gaze towards the expanded methods.

When splitting a class, most participants did not notice that the add class button added a label for a new class to which they could drag and drop methods. Most also did not find renaming classes intuitive. P2 suggested to prompt the user for a name first before moving the methods into that class. P3 mentioned that a confirmation message would be useful.

Suggesting Partition Recommendations. Task (6), where participants were asked to move a node from one class to another, was intuitive to all 8 participants since they use a similar drag and drop interaction other microservice recommendation tools. Upon dragging and dropping the class node, the suggestion box popped up at the top of the screen. Most users did not immediately notice the box. P7 mentioned that the suggestion box was not salient enough and suggested placing it next to the node that was just moved. As we prompted them to shift their focus to the suggestion box and explain its purpose, three out of the eight users (P4, P5, P7) correctly interpreted the pop up as a suggestion to move another node and highlighted the importance of this feature. P7 mentioned that *"This is good. I like this. When you are moving a class from one partition to another, if there is a dependency with another class then it probably has to be moved as well, so it makes sense. We used to do that with an application I worked with."* Despite correctly understanding the suggestion, P4 mentioned wanting to know more information about it. They stated *"For me to accept something I need to know why I’m moving it. The fact that I can accept or reject, that’s good I like that but what I wanna know why are you asking me to move it."*. Both P4 and P5 also mentioned that the node referred to in the suggestion should be highlighted on the display. 5 out of 8 users (P1, P2, P3, P6, P8) incorrectly interpreted the repartition suggestion as a confirmation of the node that they moved in task (6), despite its reference to a different class name. P2 stated while reading the suggestion "So they basically ask you for confirmation to check that you have not done it my mistake". When told that the suggestion was referring to a different node, they stated that *"If you don't know the application well enough, a lot of these things are very similar sounding. So maybe instead say do you ALSO want to move this other one to show that this is a different one"*. While this confusion could be due to the fact that the referenced node was not highlighted, it could also be due to our design choices for the repartition suggestion feature C1. P3 mentioned *"That's a good feature. We probably need to get a confirmation that the original move we made worked. And then maybe a recommendation feed. It appearing as a pop up threw me off cause that just seems like an error or a confirmation. You could have a list of recommendation on the side and call it intelligent guidance. You can see it at your own pace. I might not have time to do it on the spot, I might need to think on it. So having a list that you can pull out and tuck away with a notification that indicates "we noticed something you could do" "*. P2 and P9 also highlighted the importance of having a confirmation for moved nodes to prevent accidental interactions, a feature that does not currently exists in the applications that they are familiar with.

7 DISCUSSION

We interviewed 8 industry professionals to investigate their needs and challenges when interacting with microservice recommendation tools. Informed by our findings, we identified design considerations and developed a prototype of an interactive mixed-initiative interface that supports. We evaluated how users interacted with our UI features via semi-structured interviews. Below, we discuss some of the implications of our findings.

In our formative interviews, we asked participants about their needs and pain points when interacting with microservice recommendation tools, as well as their thoughts on the utility of AI explainability. We found that most users asserted the benefits of exploring their applications at a method-level granularity in addition to class-level, provided that the display does not become crowded and cumbersome. Users brought up concerns about the potential difficulties when navigating graphical microservice tools for large applications, and highlighted the need for more filters. Users also highlighted the utility of having more insight into the algorithm's confidence but were not enthusiastic about understand low level AI mechanics.

When evaluating our prototype, users responded positively to the several functionalities provided by the features implemented. Notably, they highlighted the importance of the ability to explore methods and their partitions, and the ability to receive partitioning recommendations. When it comes to the interpretability and usability of the features, all users correctly interpreted the class uncertainty and were able to scan the display to detect low confidence nodes and split a class. When graphically expanding classes to methods, while users correctly performed the interaction and correctly interpreted the smaller circles as methods, they missed one method that was part of another partition and less salient. We found that users did not understand partition suggestions, but instead thought that it was a confirmation of their changes. Many studies have found that people tend to misunderstand or ignore suggestions presented to them while they are conducting a task [14]. Future work needs to examine ways to integrate other important features and examine how to present suggestions to strike the right balance between encouraging their use while minimizing intrusion. Nonetheless, our work provides valuable insights into how users interact with various UI components in a mixed-initiative microservice recommendation tool.

8 LIMITATIONS & FUTURE WORK

We determined the UI features for our interface based on the needs and challenges identified by software developers that work within the same enterprise and are familiar with similar applications. While the themes that were brought up can be generalized across different use cases, they are likely anchored based on the participants' background and experience. More research needs to be conducted to examine a more diverse set of users. In addition, our work provides a preliminary examination of UI features for microservice recommendation tools and is not exhaustive. Further research is required to investigate interactions with a comprehensive tool in a more realistic settings and across a range of use cases. Finally, we encourage researchers to adopt a human-centered approach to inform the design microservice recommendation tool. While our work is a start to bridging the gap between application modernization and HCI approaches, there still remain several challenges in this space.

9 CONCLUSION

We presented results from a formative interview study of 8 software engineers who work at a large, international technology company. They highlighted their challenges and needs when it comes to the use of microservice recommendation tools which we classified into three common themes: information overload, customization and granularity and considerations for explainability. We developed an interactive mixed-initiative interface that strives to address their pain points and facilitate interactions with microservice recommendation algorithms. We evaluated our interface through a semi-structured set of tasks and identified upsides and challenges that users encountered. We then discussed the implications of our findings and discuss avenues for future research. We believe that human-centered research can help develop more optimal application modernization tools that work in collaboration with users and provide a platform to leverage the strengths of both users and AI. We encourage researchers to conduct more work to help bridge the gap between HCI and software engineering.

ACKNOWLEDGMENTS

REFERENCES

[1] Firas Al-Doghman, Nour Moustafa, Ibrahim Khalil, Zahir Tari, and Albert Zomaya. 2022. AI-enabled secure microservices in edge computing: Opportunities and challenges. *IEEE Transactions on Services Computing* (2022).

[2] Zahra Ashktorab, Michael Desmond, Josh Andres, Michael Muller, Narendra Nath Joshi, Michelle Brachman, Aabhas Sharma, Kristina Brimijoin, Qian Pan, Christine T Wolf, et al. 2021. Ai-assisted human labeling: Batching for efficiency without overreliance. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW1 (2021), 1–27.

[3] Gagan Bansal, Besmira Nushi, Ece Kamar, Eric Horvitz, and Daniel S Weld. 2021. Is the most accurate ai the best teammate? optimizing ai for teamwork. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 11405–11414.

[4] Leilani Battle, Remco Chang, and Michael Stonebraker. 2016. Dynamic prefetching of data tiles for interactive visualization. In *Proceedings of the 2016 International Conference on Management of Data*. 1363–1375.

[5] Eli T Brown, Alvitta Ottley, Helen Zhao, Quan Lin, Richard Souvenir, Alex Endert, and Remco Chang. 2014. Finding waldo: Learning about users from their interactions. *IEEE Transactions on visualization and computer graphics* 20, 12 (2014), 1663–1672.

[6] R Jordon Crouser and Remco Chang. 2012. An affordance-based framework for human computation and human-computer collaboration. *IEEE Transactions on Visualization and Computer Graphics* 18, 12 (2012), 2859–2868.

[7] Filip Dabek and Jesus J Caban. 2016. A grammar-based approach for modeling user interactions and generating suggestions during the data exploration process. *IEEE transactions on visualization and computer graphics* 23, 1 (2016), 41–50.

[8] Alex Endert, Patrick Fiaux, and Chris North. 2012. Semantic interaction for visual text analytics. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 473–482.

[9] Chen-Yuan Fan and Shang-Pin Ma. 2017. Migrating monolithic mobile application to microservice architecture: An experiment report. In *2017 ieee international conference on ai & mobile services (aims)*. IEEE, 109–112.

[10] Kevin Hu, Diana Orghian, and César Hidalgo. 2018. DIVE: A mixed-initiative system supporting integrated data exploration workflows. In *Proceedings of the workshop on human-in-the-loop data analytics*. 1–7.

[11] Anup K Kalia, Jin Xiao, Rahul Krishna, Saurabh Sinha, Maja Vukovic, and Debasish Banerjee. 2021. Mono2micro: a practical and effective tool for decomposing monolithic java applications to microservices. In *Proceedings of the 29th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*. 1214–1224.

[12] Hannah Kim, Dongjin Choi, Barry Drake, Alex Endert, and Haesun Park. 2019. TopicSifter: Interactive search space reduction through targeted topic modeling. In *2019 IEEE Conference on Visual Analytics Science and Technology (VAST)*. IEEE, 35–45.

[13] Lisa J Kirby, Evelien Boerstra, Zachary JC Anderson, and Julia Rubin. 2021. Weighing the evidence: On relationship types in microservice extraction. In *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*. IEEE, 358–368.

[14] Shayan Monadjemi, Mengtian Guo, David Gotz, Roman Garnett, and Alvitta Ottley. 2023. Human-Computer Collaboration for Visual Analytics: an Agent-based Framework. *arXiv preprint arXiv:2304.09415* (2023).

[15] Shayan Monadjemi, Sunwoo Ha, Quan Nguyen, Henry Chai, Roman Garnett, and Alvitta Ottley. 2022. Guided Data Discovery in Interactive Visualizations via Active Search. In *2022 IEEE Visualization and Visual Analytics (VIS)*. IEEE, 70–74.

[16] Rina Nakazawa, Takanori Ueda, Miki Enoki, and Hiroshi Horii. 2018. Visualization tool for designing microservices with the monolith-first approach. In *2018 IEEE Working Conference on Software Visualization (VISSOFT)*. IEEE, 32–42.

[17] Vikram Nitin, Shubhi Asthana, Baishakhi Ray, and Rahul Krishna. 2022. CARGO: ai-guided dependency analysis for migrating monolithic applications to microservices architecture. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–12.

[18] Alvitta Ottley, Roman Garnett, and Ran Wan. 2019. Follow the clicks: Learning and anticipating mouse interactions during exploratory data analysis. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 41–52.

[19] Francisco Ponce, Gastón Márquez, and Hernán Astudillo. 2019. Migrating from monolithic architecture to microservices: A Rapid Review. In *2019 38th International Conference of the Chilean Computer Science Society (SCCC)*. IEEE, 1–7.

[20] Jiao Sun, Q Vera Liao, Michael Muller, Mayank Agarwal, Stephanie Houde, Kartik Talamadupula, and Justin D Weisz. 2022. Investigating explainability of generative AI for code through scenario-based design. In *27th International Conference on Intelligent User Interfaces*. 212–228.

[21] Yingying Wang, Harshavardhan Kadiyala, and Julia Rubin. 2021. Promises and challenges of microservices: an exploratory study. *Empirical Software Engineering* 26, 4 (2021), 63.

[22] Justin D Weisz, Michael Muller, Stephanie Houde, John Richards, Steven I Ross, Fernando Martinez, Mayank Agarwal, and Kartik Talamadupula. 2021. Perfection not required? Human-AI partnerships in code translation. In *26th International Conference on Intelligent User Interfaces*. 402–412.

[23] Justin D Weisz, Michael Muller, Steven I Ross, Fernando Martinez, Stephanie Houde, Mayank Agarwal, Kartik Talamadupula, and John T Richards. 2022. Better together? an evaluation of ai-supported code translation. In *27th International Conference on Intelligent User Interfaces*. 369–391.

[24] Brian Whitworth. 2005. Polite computing. *Behaviour & Information Technology* 24, 5 (2005), 353–363.