Methods for Investigating Mental Models For Learners of APIs

Amber Horvath
Carnegie Mellon University
Pittsburgh, USA
ahorvath@cs.cmu.edu

Mariann Nagy
Google, Inc.
Seattle, USA
mknagy@google.com

Finn Voichick
Washington University in St. Louis
St. Louis, USA
fvoichick@wustl.edu

Mary Beth Kery
Brad A. Myers
Carnegie Mellon University
Pittsburgh, USA
mkery@cs.cmu.edu
bam@cs.cmu.edu

ABSTRACT

Despite almost all software development involving application programming interfaces (APIs), there is surprisingly little work on how people use APIs and how to evaluate and improve the usability of an API. One possible way of investigating the usability of APIs is through the user's mental model of the API. Through discussions with the developers and UX practitioners at Google along with our own evaluations, a distributed data processing API called Apache Beam has been identified as difficult to use and learn. In our on-going study, we investigate methods for understanding users' mental models of distributed data processing and how this understanding can lead to design insights for Beam and its documentation. We present our novel approach, which combines a background interview with two natural programming elicitation segments: the first designed for participants to express a high

CHI'19 Extended Abstracts, May 4–9, 2019, Glasgow, Scotland UK

ACM ISBN 978-1-4503-5971-9/19/05.
https://doi.org/10.1145/3290607.3312897

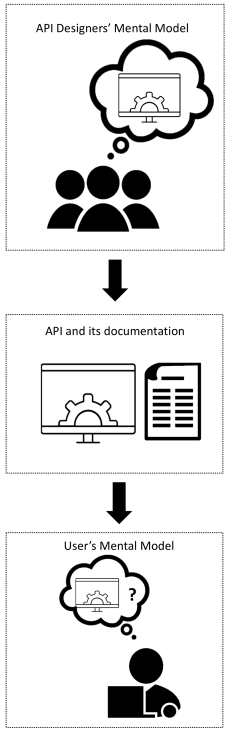

Figure 1: Designers develop a mental model of how they believe the API operates and how it should be used by a programmer. The API and its documentation, including the chosen method and class names, primary uses cases, and constraints all inform the user's mental model. Designers expect the user's mental model to be identical to their own, but, because they can only communicate through the API and its documentation, these must be designed such that the user can develop a satisfactory mental model. Adapted from [7].

level mental model of a data processing API while the second asks questions contextualized to a data processing task to see how participants apply their conceptual understanding to a more specific situation. Our method shows promise as pilot participants expressed a "dataflow" mental model that matched one way that Beam has been described, resulting in a potential design modification.

CCS CONCEPTS

• **Human-centered computing** → **User studies**.

KEYWORDS

API Usability; mental models; usability evaluation methods.

INTRODUCTION

Currently, over 20,000 APIs are registered at programmableweb.com, with countless more private APIs for internal use. The majority of programming that software engineers do every day is finding and learning these APIs and using them together to create new software projects. However, while APIs are ubiquitous, there is surprisingly little work on how to design usable APIs and how API designers may evaluate their usability prior to publishing them [6]. Some API designers are limited to releasing their APIs as their primary method of receiving feedback on their design, as they lack expertise in human-centered usability evaluation methods [5]. Poorly designed APIs may confuse the user on how to use them, which has lead to bugs and significant security issues [1].

One potential way to create more usable APIs is by scaffolding a correct *mental model* [7], the conceptual model in a user's mind of how something works. Norman states "the major clues to how things work come from their perceived structure - in particular from signifiers, affordances, constraints, and mappings" [7], an analysis which can be applied to APIs (see Figure 1). Previous work has shown that users with more accurate mental models of how a machine learning model worked were able to use the system more effectively [4]. In the context of API design, there has been little work on how an API designer can understand a user's mental model to motivate design decisions. The most relevant literature investigated how children with no programming experience create a mental model about programming constructs in the context of the Pac-Man game. Participants were given a set of scenarios and asked to describe how they would tell a computer to accomplish the action shown in the scenario. This process, called "natural programming elicitation" [8], is adapted for our methodology.

Mental model creation may be more difficult when the API is more complex. One such complex API is Apache Beam [2], an API that allows users to run and manipulate large, potentially streaming data sets using a distributed parallel processing infrastructure. Through discussions with Google's Apache Beam team along with our own analyses including writing Beam programs and administering

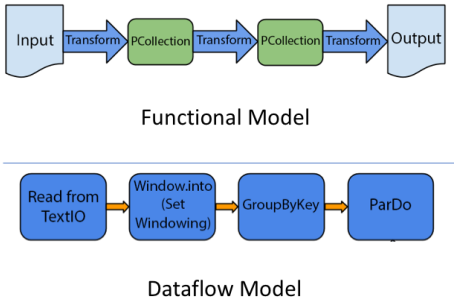

Figure 2: Two programming models which are both used in Apache Beam's actual documentation (these pictures are directly copied from the same documentation page: https://beam.apache.org/documentation/programming-guide/, © The Apache Software Foundation). The functional model has each node represent a collection of data, while the edges represent actions performed on the data. The dataflow model is the opposite with the nodes representing actions and the edges representing data.

a survey at a Beam development conference, we have found that Apache Beam is difficult to learn and use. We have multiple hypotheses as to why this is, including that the Python implementation of Beam has overloaded operators used within the API that are dissimilar to typical Python syntax, distributed data processing is challenging to understand thus the API is difficult to learn, and that the documentation supports conflicting mental models through inconsistent diagrams, shown in Figure 2. We chose to investigate the user's mental models as we believe this investigation could lead to interesting insights into how an API designer can structure the aspects of an API and its documentation by understanding what constructs, functionality and metaphors are more natural to a user. Furthermore, our collaboration with the Beam team gives us a unique opportunity to test our proposed methods and design changes.

We are interested in investigating how an API designer's understanding of a user's mental model can be used to evaluate the design of an API and propose changes (in our case, proposing changes to Apache Beam). More specifically:

- RQ1: What's an effective methodology to extract users' mental models for an API?
- RQ2: How can we use knowledge of the target users' mental models to evaluate the usability of an API and suggest modifications?

Through this work, we expect to produce a method which API designers may use when designing and evaluating their API. We are testing the proposed methodology on Apache Beam and its documentation and expect to recommend and evaluate changes.

METHOD

To investigate the questions listed above, we are conducting semi-structured interviews with users of distributed data programming APIs. In developing the method, we are taking an iterative approach to our interview questions. To help refine the questions, we specifically chose to recruit PhD students in the Human-Computer Interaction Institute at Carnegie Mellon University (CMU) who have experience both in distributed data processing and in user study and method design. Currently, we have run two participants (P1 and P2) in this method development phase. Once the questions are refined, we will begin recruiting participants from various target populations, such as students in CMU's Masters of Computational Data Science who learn distributed data processing APIs similar to Beam. We will also recruit participants, both novices and experts, with other kinds of data analysis experience but who have not used *distributed* data processing APIs, as Beam aims to be accessible to people who do not have experience with distributed computing.

Procedure. The interviewer begins by asking about the participant's background and experience using data processing APIs and their most recent project involving such an API, since this is likely to inform the mental model they hold.

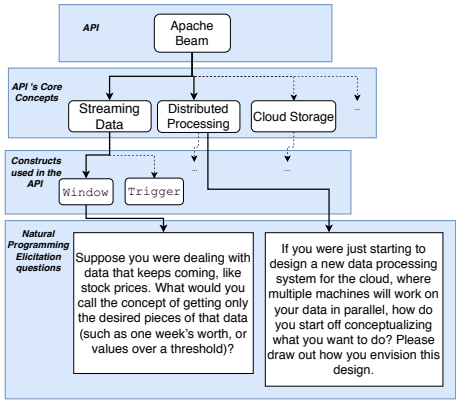

Figure 3: How the natural programming elicitation questions were generated. The root of the tree is the API itself, in our case Beam. The root has child nodes representing Beam's core concepts, including streaming data, distributed computing, and cloud storage. These core concepts were derived from the Beam documentation, along with our own usage of the API and reading Stack Overflow discussions. These concepts have child nodes representing how they currently are manifested in the API. To identify how users conceptualize the core concepts, we generate questions using the natural programming elicitation method, where we try to use prompts that do not bias participants' answers. To identify mental models, we have participants describe how they would begin designing a data processing system as this results in a high level architecture representing how a participant wants to structure their program logic. For eliciting terminology, we describe the high level operations a certain construct would perform but made sure not to include any words that would normally be considered for a function or class name.

We then conduct an adaptation of the "natural programming elicitation" method, which asks programmers to express how they think about the design of a program and the constructs used therein by asking them to write down how they would want to call the API without having a formal API specification. Structuring the questions such that they do not prime a participant to simply parrot back the terminology used in the question is difficult and requires multiple iterations to get correct, hence the iterative design. Our adaptation of natural programming elicitation involves two phases.

The first phase has the interviewer ask abstract questions to understand participants' high level comprehension of concepts related to distributed data processing. In developing these abstract questions, we focused on the concepts and constructs present in Beam, as shown in Figure 3. The first question has participants describe how they conceptualize a design for a distributed data processing task to see if participants hold a mental model similar to either one shown in Figure 2. After the first participant found our original question too difficult to answer, we modified the question to explicitly ask participants to draw out how they envision this design, resulting in the question shown in the bottom right of Figure 3. Drawing exercises have been done in prior mental model research [3]. The other questions attempt to understand what terminology participants use when referring to the API's core concepts.

The second phase of the natural programming elicitation segment asks participants to brainstorm design solutions to data processing tasks. We chose to ask these specific questions second so participants are not primed by the context of the tasks when answering the more abstract questions in phase one. The participants are told that they are not designing the API itself, but are instead writing down how they would like to *use* the API, similar to [8]. In the first task, participants are told to imagine that they are trying to sample Google's stock price once an hour for a week to compute an average, where the stock prices are continually being read by the API. Then, in the second task, participants are asked how they would tell the computer to start collecting data, how to perform some operation on the data, and how to write the data to an external storage space.

Analysis. Transcribed interview excerpts are qualitatively coded to categorize the different types of mental models users hold, using the models in Figure 2 as a basis. We also analyze the transcripts to see what terminology the participants naturally use when referring to different data processing concepts. Considering we are interested in evaluating the usability of Beam, we are interested in how well the constructs that people naturally express match the constructs supported in Beam. For example, one transformation Beam supports is merging two disparate sources of data. This operation may be called a "join" in a system such as SQL, but the participant may also refer to it as a "combine" - Beam's name for the term.

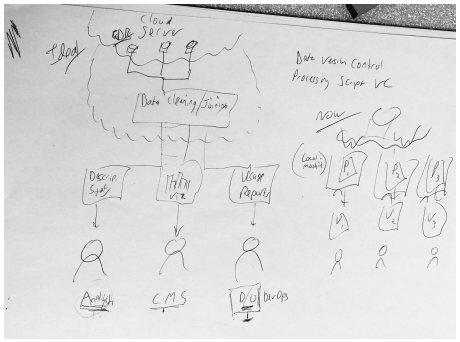

Figure 4: The second pilot participant's drawing of their data processing project. The nodes represent analyses they perform on their data (including joining, visualizations, and statistical analyses), while the edges represent the flow of the data. This matches the dataflow model.

Table 1: Constructs chosen by both participants during the natural programming elicitation section compared to the constructs available in Beam. The first row discusses operations to perform on streaming data, the second represents how to select streaming data, and then third represents how to group disparate data sources.

Beam's name for concept	P1	P2
PTransform	Apply	Transform, Normalize
Window	Filter	Batched, Snapshots
Combine, GroupByKey	Join, Merge	Join, Merge

PRELIMINARY RESULTS AND DISCUSSION

The Method. Our method has so far succeeded in eliciting a consistent *mental model* expression across our first two participants, which would have clear design implications - the Beam documentation may benefit by exclusively using the preferred model. The *terminology* used by the pilot participants has not been consistent enough to suggest any changes to Beam's design. This may be a result of the abstract natural programming elicitation questions being too broad, so further versions of the protocol will add more constraints. Alternatively, we may find that users actually differ in their preferred terminology. All of these results are still preliminary and no conclusive decisions regarding Beam's documentation should be made until more data is collected and analyzed.

Mental Models. Preliminary analysis revealed that both participants thought of their data processing tasks with a dataflow mental model. P1 stated "I guess in terms of the applications I'm used to... we might have, say, multiple machine learning or data processing nodes in that graph" implying that they interpret the data as the edges, and the nodes as where the processing occurs. Unexpectedly, this insight came out of a question attempting to understand the terminology people use when referring to applying some operation to a set of data, resulting in us modifying the protocol to allow drawing. P2 drew an image where the data processing steps are nodes, matching the dataflow model (see Figure 4). Assuming this trend continues, Beam's documentation may benefit by exclusively using the dataflow model in its graphs. However, assuming this trend does not continue and mental models are varied among participants, we may begin asking a new question such as "what mental model would be easiest to teach and most helpful to the user?".

Terminology. Preliminary analysis of the terminology the participants used in the natural programming elicitation questions showed almost no similarity to the names used for corresponding functionality in Beam (see Table 1). This suggests that perhaps Beam's function names may not match what users expect. However, the original questions were vague and did not include more complex data processing situations. As Norman stated, one way users form mental models is through constraints [7], and both participants mentioned constraints when coming up with their answers, such as performance requirements. The next iteration of our study protocol will explicitly ask about the constraints users must take into account by adding additional complexity to the questions through asking additional follow-up questions. These follow-up questions will add constraints to the data processing task.

Only P1 was able to complete the task section as P2's session was limited to 30 minutes. P1 described using a function called "listen" to collect streaming stock price data with a parameter called "until" to denote how long the listen function should remain open to new data (see Figure 5). In Beam, to collect streaming data, an object called a "window" must be instantiated and passed in as a parameter to a method of the data collection object which returns a new partitioned data collection. The windowing

Python Beam code:

```
fixed_windowed_items = (
    stockPrices | 'window' >>
    beam.WindowInto(window.
    FixedWindows(60)))
```

Java Beam code:

```
PCollection<Double> stockPrices
    = ...;
PCollection<Double>
    fixedWindowedItems =
    stockPrices.apply(Window<
    Double>.into(FixedWindows.
    of(Duration.standardMinutes
    (60))));
```

P1's imagined code:

```
stockPrices.listen(until =
    never, frequency = 60min)
```

Figure 5: Participant P1's ideal code versus what is actually supported in Beam in Python and Java for the Google stock price task given the prompt: "this data is constantly changing - write the code for how you would tell the computer program to group the data". P1 favors a less verbose approach where each parameter denotes a behavior the function reading the streaming data should perform. In contrast, the functionality in Beam requires a separate "apply" function be called on the data set and create a separate data collection for each hour.

mechanism forces the creation of a new collection object and requires many nested function calls. This result is further supported by our previously administered survey - some users stated that "windowing" is not intuitive.

CONCLUSION

Understanding the mental models held by users of complex systems, such as distributed data processing APIs, can reveal interesting insights into the assumptions users may hold and how small changes to the constructs, names, and documentation may aid in the usability of the API. While further testing must be done, the initial results appear promising in uncovering how people think about data processing. This method could be applied to other APIs to uncover places where the target API users may be thinking about concepts in a different way than the designers, and thereby inform where the design may change.

ACKNOWLEDGMENTS

This research was funded in part through a grant from Google and in part by NSF IIS-1827385. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsors.

REFERENCES

[1] Sascha Fahl, Marian Harbech, Perl Henning, Markus Koetter, and Matthew Smith. 2013. Rethinking SSL Development in an Appified World. In *Proceedings of the 2013 ACM SIGSAC conference on Computer and Communications Security (CCS '13)*. ACM, New York, NY, USA, 49–60. https://doi.org/10.1145/2508859.2516655

[2] The Apache Software Foundation. 2018. Apache Beam: An advanced unified programming model. beam.apache.org. Accessed: 2019-01-02.

[3] Ruogu Kang, Laura Dabbish, Nathaniel Fruchter, and Sara Kiesler. 2015. "My data just goes everywhere": user mental models of the internet and implications for privacy and security. In *Proceedings of the Eleventh USENIX Conference on Usable Privacy and Security (SOUPS '15)*. USENIX Association, Berkeley, CA, USA, 39–52.

[4] Todd Kulesza, Simone Stumpf, Margaret Burnett, and Irwin Kwan. 2012. Tell me more?: The Effects of Mental Model Soundness on Personalizing an Intelligent Agent. In *ACM Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 1–10. https://doi.org/10.1145/2207676.2207678

[5] Lauren Murphy, Mary Beth Kery, Oluwatosin Alliyu, Andrew Macvean, and Brad A. Myers. 2018. API Designers in the Field: Design Practices and Challenges for Creating Usable APIs. In *Symposium on Visual Languages and Human-Centric Computing (VLHCC '18)*. IEEE, 249–258.

[6] Brad A. Myers and Jeffrey Stylos. 2016. Improving API usability. *Commun. ACM* 59, 6 (2016), 62–69. https://doi.org/10.1145/2896587

[7] Don Norman. 2013. *The Design of Everyday Things*. Basic Books, New York, NY, USA, 26, 31.

[8] John F. Pane, Chotirat "Ann" Ratanamahatana, and Brad A. Myers. 2001. Studying the language and structure in non-programmers' solutions to programming problems. *International Journal of Human-Computer Studies* 54, 2 (2001), 237–264.