Towards a Model of Library Use

Ava Heinonen
Department of Computer Science
Aalto University
Ava.Heinonen@aalto.fi

Abstract

Programmers often struggle when using libraries because of difficulties in understanding how the library can be used to achieve their desired outcome. Much of the existing literature has focused on API usability and documentation. However, limited research has been done to gain insight into the processes we seek to support with documentation and usable APIs — the processes of understanding and using libraries.

In this work-in-progress paper, we present initial results of a study examining programmers' cognitive processes and mental model development as they refactor an open-source web application to use a new library. We discuss the analysis of the pilot protocol, and the initial insights gained through this analysis. The initial insights suggest that developers form understanding of the library and the solution not only through seeking information but also through interaction with the library by implementing and testing code.

1. Introduction

Software libraries can be difficult to use (Robillard & DeLine, 2011; Samudio & LaToza, 2022). One study on the problems programmers face when developing web applications indicated that many of the problems were caused by programmers struggling to understand how to use libraries and debug solutions implemented using libraries (Samudio & LaToza, 2022). Studies on library-related questions and help requests from programmers show that these problems are not limited to web programming (Wang & Godfrey, 2013; Hou & Li, 2011). As modern software development is highly based on the use of libraries and other reusable assets, finding ways to help programmers understand how to use libraries becomes increasingly important (Taivalsaari, Mikkonen, & Mäkitalo, 2019).

Research on learning and using libraries has focused on API usability and documentation — finding solutions for the problem of understanding and using libraries. Studies have sought assessed programmer difficulties with API documentation (Robillard & DeLine, 2011), evaluated ways to create effective documentation (Meng, Steinhardt, & Schubert, 2020), and developed ways to assess API usability (Piccioni, Furia, & Meyer, 2013). However, limited research has been conducted to analyze the processes we seek to support with API design and documentation — the problem of understanding and using libraries.

In recent years there have been efforts to bridge this gap in the literature. Studies have sought to model and analyze programmers' information seeking behavior when developing solutions using an unfamiliar library (Kelleher & Ichinco, 2019; Kelleher & Brachman, 2023; Sparmann & Schulte, 2023). One approach was the COIL model, that sought to model programmers' information seeking behavior when learning to understand new libraries (Kelleher & Ichinco, 2019; Sparmann & Schulte, 2023). The model describes three stages of the library use process: Information collection stage consisting of seeking and acquiring relevant information from online information sources, Information organization stage where the acquired information is organized to form a solution, and solution testing stage where the solution is implemented and tested (Kelleher & Ichinco, 2019). Another approach was the utilization of data-frame theory of sensemaking to investigate the cognitive processes underlying programmers information seeking (Kelleher & Brachman, 2023). Focusing on online information seeking behavior, they analyzed programmers search terms and navigation behavior to identify different stages in the sensemaking process (Kelleher & Brachman, 2023). These studies bring insight into developer behavior when developing the required understanding to implement solutions using unfamiliar libraries. However,

these studies do not aim to analyze the motivations underlying the behavior. Therefore, the question of what programmers seek to achieve, and what is the understanding they seek to form to achieve it remains.

While the recent studies have focused on programmer behavior, literature from the early years of software libraries and software reuse developed theories on the tasks of using libraries and the understanding required to conduct these tasks (Krueger, 1992; Détienne, 2002; Fischer, 1987). These studies identified what a programmers has to do to successfully use a library. They theorize, that library use would include locating and selecting suitable artifacts to use (Krueger, 1992), coordinating artifacts to form solutions (Fischer, 1987), modifying artifact behavior through arguments to achieve the desired behavior (Krueger, 1992; Détienne, 2002), and integrating artifacts into the program code (Krueger, 1992). Early literature on reuse also theorizes about the types of knowledge programmers would need to conduct reuse tasks (Fischer, 1987; Krueger, 1992). This includes knowledge about the library, the artifacts it provides, and how those artifacts can be configured into solutions (Krueger, 1992; Fischer, 1987). It also includes knowledge about the function and interface of the individual artifacts (Krueger, 1992; Fischer, 1987). While it is quite possible that these theories apply to modern software reuse as well, technology has rapidly changed in the last 30 years. Therefore, assessing the degree to which these theories hold in modern programming environments is required.

In this paper, we discuss an ongoing study investigating the tasks of library use in modern software development environments. The study aims to analyze the tasks involved in using and selecting libraries, and the understanding required to complete these tasks. Specifically we seek to answer the following research questions:

RQ1 What are the tasks included in using and selecting libraries?

RQ2 What do developers need to understand about the library and the task situation to conduct these tasks?

RQ3 How do developers form this understanding?

- RQ3.1 What information do developers seek?
- RQ3.2 What activities do programmers conduct to acquire and process information?

RQ4 How do programmers move between the tasks during a library use situation?

To this end, we analyze the think-aloud protocols of professional developers learning to use a library for a refactoring task. In this work-in-progress paper, we discuss the study methodology and the ongoing data analysis process. Furthermore, we discuss our initial observations.

2. Methodology

We used the think-aloud method to study developers' cognitive processes when refactoring an open-source web application to use a new library. The think-aloud method is a well-established technique used to study cognitive processes in various fields (Ericsson & Simon, 1980; Panadero, Pinedo, & Ruiz, 2025).

2.1. Study Task and Study Process

In the study, participants worked on refactoring the front-end of an open source Web Application Habitica[1] to use a new time and date library instead of Moment.js. Each participant had one hour to work on selecting a library to use and beginning to refactor the Habitica front-end. The participants were assured that they were not expected to complete the task. Participants were allowed to select which parts of the front-end code to refactor. However, they were instructed that one possible starting point could be the calculateTimeTillDue() method in the task.vue file. This was done so that participants could focus on the selection and refactoring tasks and not on deciding on a suitable starting point.

[1] https://github.com/HabitRPG/habitica

Participants were provided a laptop. The laptop had Visual Studio Code IDE with the Habitica source code already open. Habitica was already running in the background and the participants could view the Habitica application in the browser. The browser window with Habitica had developer tools already open. In another browser tab, we had a list of potential Moment replacements provided by the Moment team open [2].

Before the study, participants were asked to respond to a background survey. The survey contained eight questions about the participant's employment, development experience, and familiarity with the technologies used in the study.

The study procedure was divided into two phases. In the first phase, participants were informed about the study and the study task. Participants received verbal and textual descriptions of the study task and the opportunity to ask questions. Participants were also instructed about think-aloud.

In the second phase, participants worked on a refactoring task. The laptop screen was recorded, the participant was filmed, and their voice was recorded during the task. If the participant requested not to be filmed, only their vocalizations were recorded.

A researcher sat with the participant, making notes, and asking the participant to vocalize their thoughts if they were silent for more than 30 seconds.

2.2. Participants

Twelve participants participated in the data collection. All participants were, at the time of data collection, employed in software development roles. All participants had some experience with JavaScript and web development.

Participants were allowed to speak either Finnish or English during the think-aloud. All participants were required to have good enough English or Finnish skills to communicate their thoughts in either language. Two of the participants spoke Finnish, and eight spoke English. However, from the English speakers all spoke English as a second language.

Due to technical difficulties, the data collected from two participants could not be used. In these two cases, no sound was recorded, and thus the think-aloud data was lost. Therefore, the final data set contains data from 10 participants.

Participants were recruited through posters and contacts in companies and other organizations. Participants received a 10 euro gift card to a coffee shop for participation.

2.3. Data Analysis

Data analysis is currently in progress. We use qualitative coding to analyze the think-aloud protocols. The protocols were first segmented into segments corresponding to one *activity*. Then a codebook was developed using existing literature on library learning and use, and the analysis of a pilot protocol. The protocols were then coded, while iteratively improving on the codebook as new topics emerge from the data.

2.3.1. Segmentation

First, after the think-aloud protocols had been transcribed, the protocols were segmented into segments corresponding with a an *activity*, a coherent set of actions that achieve one goal. For example, one activity could be writing a piece of code, information seeking activity such as searching or reading, or cognitive activity such as formulating a plan.

2.3.2. Code Book and Code Book Development

Initial codebook was developed using existing literature on library learning and use, and the analysis of a pilot protocol. The initial codebook was then tested by applying it to protocols, and further revised. The initial codebook was then iteratively revised throughout the coding process.

The initial codebook contains the following codes: 1) Activities, 2) Information needs, 3) Mental Mod-

[2] https://blog.logrocket.com/5-alternatives-moment-js-internationalizing-dates/

els, 4) Tasks, and 5) Stages.

Activities refer to activities conducted by the participant. These are identified based on both the transcript and the screen recording.

Information needs refer to the information participant is seeking to acquire. For example, in the quotation below participant seeks information about the return value of an API method:

> "**. . . What does this return? The diff function...**
>
> . . . Maybe it already returns the duration.
>
> Let's see.
>
> [...] . . . Okay, so it already returns a duration"

Mental Models refer to participant's understanding of some aspect of the library or the task and the task context. For example, in the quotation below participant utterances indicate their understanding of what an API method does:

> "Returns the end of date for the given date.
>
> **Okay, so it will also change it to just different time, but the same date.**"

Tasks refer to the task participant is working on. We used existing literature on library use to identify an initial set of tasks we expected to encounter in the data. This list was further refined as more data was analyzed. For example, in the quotation below participant identifies an API method that they believe they can use in their solution:

> "...so now I'm looking if they have like a function that is named in a similar way, because it's like a pretty intuitive way to name a function.
>
> So I search,uh, subtract.
>
> The defined number of seconds from the given date.
>
> So it looks like this is this function."

Stages refer to the stage of the task. During the analysis of the pilot protocol, we identified four stages of a refactoring task. These were: 1) Understanding the goal, where participant forms good enough understanding of the functionality to be refactored to start refactoring it, 2) Solution Design, where participant forms an understanding of how the desired functionality can be achieved using their selected library, 3) Solution Implementation, where participant implements the solution, and 4) Solution evaluation where the participant seeks to evaluate to what extent the implemented solution achieves their goals. For example, the below quotation is from an evaluation stage where the participant prints the value returned by an API call they had implemented to verify that it works as they intended it to:

> "So, I'm just going to paste this here, just to see if it prints falses in the console.
>
> . . . date is false which okay good I consider this a success"

2.3.3. Current Status

Data analysis is currently ongoing. The first round of coding has been conducted using the initial codebook. The codes from the first round are currently being analyzed and consolidated to develop the final codebook.

After the final codebook has been developed, a second round of coding will be conducted using the final codebook.

2.4. Initial Insights

The first round of coding has allowed us to develop some initial insights. In contrast to the COIL model that highlights the importance of online information sources in learning to use libraries (Kelleher & Ichinco, 2019), our data seems to indicate that programmers also learn to understand how artifacts work and how they can be used by trial and error and by observing their behavior in a program. Multiple participants indicated, that as they were implementing code they were trying things out to see how it works or if it would work. One participant even commented on this stating *"I find it easier to try things out rather than think"* as they were testing which way around they should order two API methods.

Participants also indicated, that using artifacts allowed them to test their hypotheses about the artifacts and their use. For example, one participant was trying to figure out the arguments to give to an API method by reading API documentation, and stated: *"ok anyway, I think if I don't import and don't test it I don't get any clue that my understanding is correct from this"*.

Our current hypothesis is, that interaction with the IDE plays an important role in learning to understand how library artifacts work and can be used. It seems to be used for hypothesis verification by testing out hypotheses of artifacts and their use. It may also be used for gaining insight into code behavior when the programmer does not have good enough mental models to be able to mentally simulate code execution.

This would be in contrast with prior literature on API learning (Kelleher & Ichinco, 2019; Sparmann & Schulte, 2023; Kelleher & Brachman, 2023), in which learning is assumed to occur primarily through reading online information sources. This would also indicate avenues for supporting learning to understand libraries through IDE tools.

3. References

Détienne, F. (2002). Software reuse. In *Software design—cognitive aspects* (pp. 43–55). Springer.

Ericsson, K. A., & Simon, H. A. (1980). Verbal reports as data. *Psychological review*, *87*(3), 215.

Fischer, G. (1987). Cognitive view of reuse and redesign. *IEEE Software*, *4*(4), 60.

Hou, D., & Li, L. (2011). Obstacles in using frameworks and apis: An exploratory study of programmers' newsgroup discussions. In *2011 ieee 19th international conference on program comprehension* (pp. 91–100).

Kelleher, C., & Brachman, M. (2023). A sensemaking analysis of api learning using react. *Journal of Computer Languages*, *74*, 101189.

Kelleher, C., & Ichinco, M. (2019). Towards a model of api learning. In *2019 ieee symposium on visual languages and human-centric computing (vl/hcc)* (pp. 163–168).

Krueger, C. W. (1992). Software reuse. *ACM Computing Surveys (CSUR)*, *24*(2), 131–183.

Meng, M., Steinhardt, S. M., & Schubert, A. (2020). Optimizing api documentation: Some guidelines and effects. In *Proceedings of the 38th acm international conference on design of communication* (pp. 1–11).

Panadero, E., Pinedo, L., & Ruiz, J. F. (2025). Unleashing think-aloud data to investigate self-assessment: Quantitative and qualitative approaches. *Learning and Instruction*, *95*, 102031.

Piccioni, M., Furia, C. A., & Meyer, B. (2013). An empirical study of api usability. In *2013 acm/ieee international symposium on empirical software engineering and measurement* (pp. 5–14).

Robillard, M. P., & DeLine, R. (2011). A field study of api learning obstacles. *Empirical Software Engineering*, *16*, 703–732.

Samudio, D. I., & LaToza, T. D. (2022). Barriers in front-end web development. In *2022 ieee symposium on visual languages and human-centric computing (vl/hcc)* (pp. 1–11).

Sparmann, S., & Schulte, C. (2023). Analysing the api learning process through the use of eye tracking. In *Proceedings of the 2023 symposium on eye tracking research and applications* (pp. 1–6).

Taivalsaari, A., Mikkonen, T., & Mäkitalo, N. (2019). Programming the tip of the iceberg: software reuse in the 21st century. In *2019 45th euromicro conference on software engineering and advanced applications (seaa)* (pp. 108–112).

Wang, W., & Godfrey, M. W. (2013). Detecting api usage obstacles: A study of ios and android developer questions. In *2013 10th working conference on mining software repositories (msr)* (pp. 61–64).