TrackThinkDashboard: Understanding Student Self-Regulated Learning in Programming Study

Ko Watanabe [1], Yuki Matsuda [2], Yugo Nakamura [3],
Yutaka Arakawa [4], and Shoya Ishimaru [5]
[1] German Research Center of Artificial Intelligence (DFKI GmbH),
[2] Okayama University, [34] Kyushu University, [5] Osaka Metropolitan University

Abstract

In programming education, fostering self-regulated learning (SRL) skills is essential for both students and teachers. This paper introduces *TrackThinkDashboard*, an application designed to visualize the learning workflow by combining web browsing and programming logs in one unified view. The system aims to (1) help students monitor and reflect on their problem-solving processes, identify knowledge gaps, and cultivate effective SRL strategies, and (2) enable teachers to identify at-risk learners more effectively and provide targeted, data-driven guidance. We conducted a study with 33 participants (32 male, one female) from Japanese universities—some with prior programming instruction and some without—to explore differences in web browsing and coding patterns. The dashboards revealed multiple learning approaches (e.g., try-and-error, try-and-search, and more) and highlighted how domain knowledge influenced overall activity flow. We discuss how this visualization can be used continuously or in one-off experiments, the privacy considerations involved, and opportunities for expanding data sources for richer behavioral insights.

1 Introduction

Programming education has emerged as a vital component of modern professional skill development. To meet the demands of an increasingly technology-driven world, educators must provide hands-on learning experiences that nurture students' self-regulated learning (SRL) skills [30]. SRL empowers learners to take charge of their educational journey through

[1]ko.watanabe@dfki.de
[2]yukimat@okayama-u.ac.jp
[3]y-nakamura@ait.kyushu-u.ac.jp
[4]arakawa@ait.kyushu-u.ac.jp
[5]ishimaru@omu.ac.jp

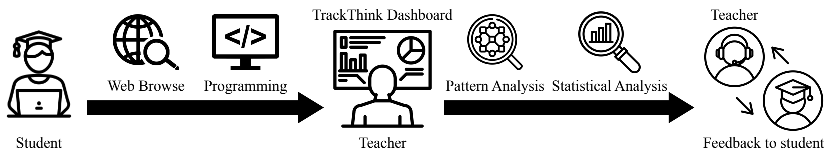


Figure 1: Overall workflow of *TrackThinkDashboard*.

goal setting, self-monitoring, reflection, and refinement, ultimately enhancing both academic performance and long-term knowledge retention.

To enhance SRL in programming education, we present the *TrackThinkDashboard*, an innovative application designed to visualize SRL activity behaviors. The application aims to empower students and teachers by integrating web browsing and programming activities into a unified view. Prather et al. mentioned that many novices do not have well-developed content knowledge or cognitive control in programming, so they lack even a basic understanding of their progress through the programming problem-solving process. Visualizing whole programming and web browsing activities can help students gain deeper insights into their learning workflows and decision-making processes, fostering a more self-aware and practical approach to learning [19].

While previous systems like SearchBar [14], popHistory [5], and *TrackThinkTS* [12] have focused on collecting web browsing logs, and tools such as Log++ [13] and Projection Boxes [10] have captured programming activity logs, these studies have not addressed the integration and classification of behaviors across both domains. We chose to combine web browsing and programming logs because, in many programming tasks, acquiring knowledge (through web searches, documentation, or Q&A sites) is tightly coupled with the application of knowledge (writing, compiling, and revising code) [19]. Observations in programming education [4, 19, 25] confirm that novice and intermediate learners rely heavily on real-time web searches, especially for syntax help or debugging. By syncing these logs in a single visualization, we give both students and teachers a complete picture of the problem-solving loop—making it clear when a student repeatedly returns to the same resource or whether they compile the code multiple times before searching.

The application serves both students and teachers by tracking and visualizing transitions between web searches and coding tasks. For students, these visualizations facilitate meta-cognitive awareness [11], enabling them to pinpoint knowledge gaps, optimize resource usage, and develop more structured SRL strategies. In parallel, the dashboard equips teachers with detailed insights into student behavior, such as prolonged resource searching, recurring coding errors, or inefficient task-switching—allowing for targeted, data-driven interventions. The platform fosters collaboration and strengthens the overall learning experience for both parties by prompting discussions around specific learning behaviors and strategies.

This paper proposes an application that leverages web browsing and programming activities to visualize students' SRL workflow in programming learning. As shown in Figure 1, we use a web browsing activity logger [12] and a programming IDE to collect logs. The *TrackThinkDashboard* synchronously visualizes two activities, offering an intuitive GUI (Graphical User Interface) that allows users to visualize their activities as a structured flowchart. Our contributions are as follows:

1. **A unified visualization of web browsing and programming**: We introduce an application that combines web browsing and programming activities as an all-in-one flowchart.
2. **We discover the pattern of SRL behavior through web browsing and programming activity**: Using our application, we identify how students transition between web browsing and programming tasks, revealing an understanding of the patterns of SRL workflow.

2 Related Work

2.1 SRL in Programming

SRL is the cyclical process of planning, monitoring, and evaluating one's cognitive, meta-cognitive, and behavioral strategies to achieve learning goals [30]. In programming, SRL encompasses how learners anticipate and design solutions, continuously monitor and debug their code, and then reflect on errors and outcomes to refine their mental models. Research by Prather et al. [19] indicates that many novices struggle to gauge their progress, suggesting that visualizing one's workflow—such as errors, code searches, and version history—can reinforce meta-cognitive skills and help students develop more accurate mental models of programming.

More recently, several studies have shown how SRL and meta-cognition theories can be applied specifically to programming courses. For instance, Loksa et al. [11] present a systematic overview of SRL frameworks, illustrating how they inform both research and practice. Meanwhile, Silva et al. [21] identify a broad spectrum of regulatory strategies that learners employ (e.g., time management, motivation, and planning), offering new insights into how SRL unfolds in the process of writing code. Collectively, these findings highlight the most common pitfalls novices face—such as challenges in debugging and sustaining motivation—and propose domain-specific scaffolding to strengthen students' self-monitoring and self-evaluation skills in programming.

In summary, building on these studies, we argue that cultivating SRL in programming should focus on enhancing meta-cognitive awareness of each learner's workflow, including how they write and troubleshoot code, as well as how they search for information. By visualizing the entire process—from planning and coding to reflection and debugging—novices can more effectively self-monitor and self-evaluate, ultimately becoming more self-regulated and proficient programmers.

2.2 Web Browsing Activity Log Collection

Several tools have been developed to log and analyze web browsing activities. Mermite [28] lets users store multiple web resources and create mashups without needing programming skills. Users can organize content, like flight prices or publication years, in any order. SearchBar [14] saves browsing and query histories along with user ratings and notes, making it easier to highlight important actions, such as favorite websites or frequently used queries. This approach helps users visualize their browsing behavior and even share patterns with others. PopHistory [5] presents web history as a bubble chart, where bubble sizes indicate frequently visited sites. LogCanvas [29] collects web search histories and visualizes them as knowledge graphs, helping users see connections between pages rather than simply listing URLs. Lastly, *TrackThinkTS* [12, 27] is a browser extension inspired by TrackThink [15]. It logs tab and window operations with timestamps and exports data in CSV format, simplifying log analysis. These tools underscore the value of web browsing logs for both personal and shared insights.

2.3 Programming Activity Log Collection

Several tools have been developed to log and analyze programming activities. Log++ [13] captures logging results in JSON format, focusing on optimizing logging statements rather than recording compilation histories or results. Projection Boxes [10] provides a summary of compilation results and timestamps, displaying code statements in a box format, but it's not designed to collect detailed compile logs. *C2Room* is an online programming IDE that logs timestamps, compiled code, and results. It also allows users to set programming tasks, recording task IDs and timestamps, making it easy to track code progress and outcomes. Similarly, WEVL [23] supports online programming environments with comparable functionality. Lastly, Log-it [9] is a Visual Studio Code extension that visualizes programming workflows, enabling users to easily reference historical code patterns through its visual interface.

2.4 Visualize Web Browsing and Programming

The integration of web browsing and programming logs has drawn attention in recent research to better understand data interactions. Prompter [17] is an IDE plugin that retrieves relevant Stack Overflow discussions, ranks their relevance using a multifaceted model, and displays them to developers within the Eclipse IDE. A study involving 33 developers reported a 74% positive response rate [17]. This concept was later expanded in Libra [18], which provides comprehensive web resource recommendations directly within the IDE. CrossRec [16] recommends third-party libraries by analyzing attached code. Brandt et al. explored how developers use online resources for technical problem-solving [4], revealing that programmers engage in *just-in-time* learning and use online materials for both acquiring new skills and clarifying existing knowledge. Additionally, Watanabe et al. investigated the estimation of programming domain knowledge from web and programming log data [25]. By categorizing participants as domain experts or novices, the study achieved a prediction accuracy of 0.95 using Random Forest, highlighting significant behavioral differences between novice and expert programmers in web browsing and coding.

Table 1: Web browsing log collected by *TrackThinkTS*.

Column Name	Description
UserID	Participant's user ID.
UserAction	User action (e.g., tab, scroll, copy).
date	Log timestamp in UNIXTIME.
Tab_URL	URL of the accessed page.
Tab_Title	Title of the accessed page.
Tab_BodyText	Body text of the accessed page.
ClipboardCopy	Copied text content.
Scroll_YAxisSpeed	Vertical scroll speed.
Scroll_VisibleText	Visible text after scroll.
Scroll_ViewPort_XScroll	Horizontal scroll viewport.
Scroll_ViewPort_YScroll	Vertical scroll viewport.
Scroll_XScrollRate	Horizontal scroll percentage.
Scroll_YScrollRate	Vertical scroll percentage.
ViewPortWidth	Viewport width.
ViewPortHeight	Viewport height.
DocWidth	Document width.
DocHeight	Document height.

Table 2: Programming log collected by *C2Room*.

Column Name	Description
time	Log timestamp in JST.
uid	User ID of the participant.
classID	Virtual room ID created by the organizer.
taskID	Task/question ID created by the organizer.
lang	Programming language used.
op	User action (e.g., compile, submit).
msg	Compile message (e.g., status, output, errors).

In summary, while existing works provide valuable insights into integrating web browsing and programming logs, none offer a clear visualization of the learning path in programming.

3 Methodology

We use two applications for logging web browsing and programming activity to visualize both activities synchronously. This section explains data source, fusion, and visualization approaches for understanding SRL.

3.1 Data Source

In this section, we explain the data source. In this study, we use web browsing and programming logs as a data source. We use the web browser

Table 3: Selected log after from *TrackThinkTS* and *C2Room*.

Column Name	Description
timestamp	Time of the action in UNIXTIME.
userID	Participant ID during SRL.
taskID	Task/question ID created by the organizer.
userAction	Action logs collected from data sources.
tabURL	URL of the accessed web page.
clipboardCopy	Text copied to the clipboard.
msg	Compile message (e.g., status, output, errors).

logger *TrackThinkTS* and *C2Room* [6], an online programming IDE for collecting programming activity logs. We will explain web browsing and programming logs in detail.

Web browsing activities are recorded through a web browser extension called *TrackThinkTS*. Table 1 shows overall logs collected by the application. One notable advantage of *TrackThinkTS* over other web browser loggers is its user-friendly interface, allowing participants to delete irrelevant logs collected during the experiment. Users need to install this extension in their web browser to utilize it. In our study, we have selected Google Chrome [7] as the preferred web browser.

Programming activities are recorded through an online web IDE called *C2Room*. Table 2 shows overall logs collected by the application. The application is tailored for online programming classes in educational institutions and companies. The log lets us understand how and when users execute their code in the compiler and the compiled result. The application also allows users to track when they are satisfied with their written code and proceed to submit it.

3.2 Data Fusion

Data fusion is applied to the collected data sources. The data fusion process comprises two primary procedures: data shaping and filtering. Data shaping involves transforming raw data into a unified format suitable for combining various data sources. On the other hand, data filtering entails selecting relevant data points following the conversion and concatenation. We will now provide a detailed explanation of each procedure.

Data Shaping: We employ data shaping as a preprocessing step to facilitate the concatenation and filtering of logs. We rename the columns for each web browsing and programming log. Specifically, we begin by renaming the columns *date* and *time* to *timestamp*. Additionally, we rename *UserID* and *uid* to *userID*. For *UserAction* and *op*, we change to *userAction*. This column renaming process aims to align the corresponding information between the two applications. Once the column renaming is complete, we convert the *timestamp* values to UNIXTIME and sort them chronologically.

[6]`https://C2Room.jp/`
[7]`https://www.google.com/chrome/`

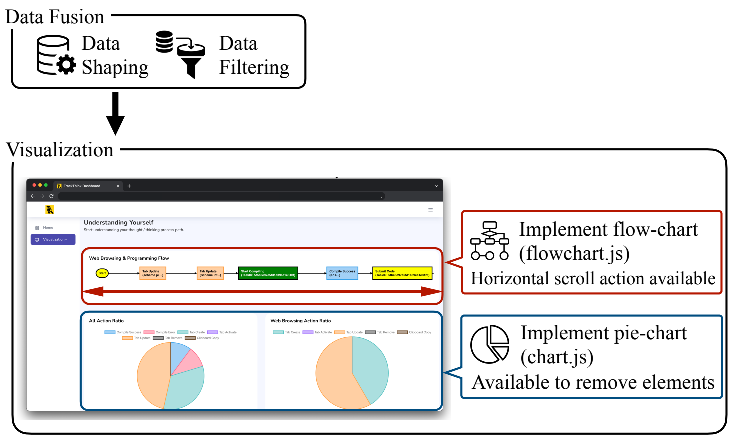

Data Fusion
Data Shaping
Data Filtering
Visualization
Implement flow-chart (flowchart.js)
Horizontal scroll action available
Implement pie-chart (chart.js)
Available to remove elements

Figure 2: Application workflow. Visualization of flowchart and pie-chart.

Data Filtering: Table 3 shows the table after data filtering. Some information is removed, such as window scroll speed from the web browsing log, class ID from the programming log, or logs involving NAN values. In order to sort logs into time order, we convert all units of the timestamp into UNIXTIME. Specifically, the programming logs collected by *C2Room* were converted from JST to UNIXTIME. All the logs are concatenated and sorted in a time order using *timestamp*.

3.3 Visualize Web Browsing and Programming

Visualization of web browsing and programming activity is performed after data fusion. Figure 2 shows the library used for visualization. For the visualization, we choose a flowchart and pie-chart. We will explain the implementation process and the reason for the choice in detail.

The flowchart is selected to visualize the problem-solving progress in a time series. The approach uses flowcharts to understand what search results participants used to arrive at their answers and what compilation errors they encountered when re-running their searches. The flowchart is implemented using *flowchart.js* [8]. It is a JavaScript library for flowchart SVG (Scalable Vector Graphics) rendering that runs in the terminal and browser. We categorize users' activities in different colors and shapes for start and stop edges. The workflow is not visualized fully on the screen, but the user can scroll horizontally to see the actions between the start and end edges. For edges like *tab activate* or *tab update*, the hyperlink is set so that users can jump to the webpage once they tap the edge.

The pie-chart is selected for visualization because the activity ratio of user action is essential in identifying users' domain knowledge [25]. The pie chart is implemented using *chart.js* [6]. Clicking each element removes a specific activity from the pie chart. This option helps teachers

[8]`http://adrai.github.io/flowchart.js`

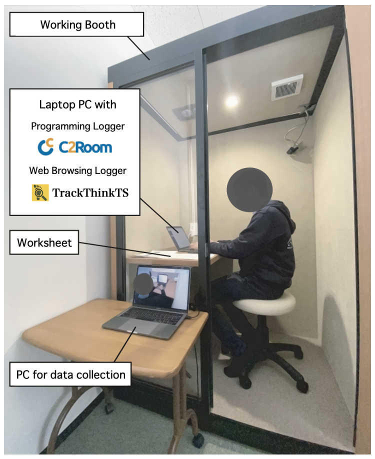


Figure 3: Experimental settings. Collect logs while solving scheme questions.

or students focus on the ratio of an activity they want to compare. It is a JavaScript library for making HTML-based charts.

4 Data Collection

4.1 Participants

In this experiment, we collect logs from lecture students (Dataset A) and non-lecture students (Dataset B). The total number of participants is 33 unique (32 males and one female) university students in Japan.

Dataset A – University students attending lectures: We collect data from 13 unique university students (12 males and one female) in Japan. Participants have taken university courses in Scheme programming language, so they have knowledge on Scheme grammar or syntax.

Dataset B – University students not attending lectures: We collected data from 20 male university students in Japan. Participants did not take any university courses related to the Scheme, so they do not have any knowledge from the class, such as Scheme grammar or syntax.

4.2 Experiment Procedure

Figure 3 shows the condition of the experiment. Before the experiment, *C2Room* and *TrackThinkTS* were installed on each participant's laptop. Under these conditions, the experiment was conducted using the following procedure: (1) The experiment conductor presents the purpose of the experiment, experimental conditions, and tools that will be provided to the participant. Only the person who agrees with the conditions can participate in the experiment. (2) The participant enters the personal workstation and starts web browsing and programming loggers. (3) Participants work on solving problems with a given schema using the *C2Room*

Table 4: Scheme Questions

ID	Question
A	Define variable *PI* as 3.14.
B	Write a scheme to show PI $*5^2$.
C	Write a scheme to show $(-b+\sqrt{b^2-3ac})/3a$.
D	Define function *areaDisk* to calculate circle area from radius r.
E	Define function *areaRing* to calculate circle area from outer and inner diameter D, d.
F	Define function *d2y* that convert US currency dollar d to Japanese currency yen. Note that 1 US dollar is 108.43 yen.
G	Define function *e2d* that convert. European currency e to United states dollar. Note that 1 euro is 1.1069 US dollar.
H	Define function *p2e* that convert British currency pond p to European currency. Note that 1 pond is 1.1632 euro.
I	Define function *p2y* that convert British currency pond p to Japanese currency. Use *d2y*, *e2d*, and *p2e* in the previous questions.
J	Define function *c2f* that convert Celsius C to Fahrenheit. Note that $f = 1.8c + 32$.

programming editor. Table 4 shows the questions for participant to solve. The order of question-solving is not restricted, but it is assumed that the participant will solve the easy questions step by step. *C2Room* will record the compiled results for each question. (4) Participants may use a web browsing to find the answer. *TrackThinkTS* will track web browsing behavior. (5) The participant submits the answer and moves on to the next question when satisfied with the code compilation. Allow the student to return to previous questions to change the answer. (6) When all questions have been answered or an hour has passed, we stop participants from problem-solving. (7) Participants remove privacy-sensitive logs from *TrackThinkTS*. Privacy-sensitive data refers to data stored during the experiment that is not related to the programming task or that the user does not want to share. Once all recorded logs are submitted to the experimenter, the participant leaves the personal workspace.

In this study, we recruit the same Scheme questions as Watanabe et al. [25]. We have prepared ten questions in order of difficulty. Easy questions are those that require fewer lines of code to solve. In this experiment, we chose Scheme (Racket), one of the dialects of LISP languages, as the programming language of the task [1]. We chose this task because of its simple language specification, used in programming courses at several universities. Also, it is tailored for lecture use, so its language specification is usually unknown to students except those attending the lecture.

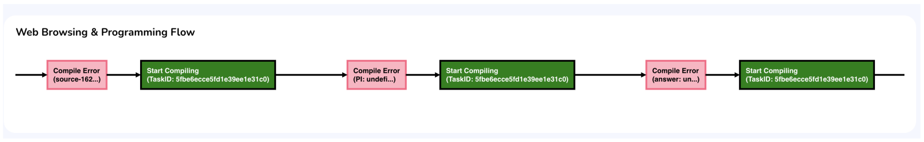

Web Browsing & Programming Flow
Compile Error (source-162...)
Start Compiling (TaskID: 5fbe6ecce5fd1e39ee1e31c0)
Compile Error (PI: undefi...)
Start Compiling (TaskID: 5fbe6ecce5fd1e39ee1e31c0)
Compile Error (answer: un...)
Start Compiling (TaskID: 5fbe6ecce5fd1e39ee1e31c0)

Figure 4: **Try-and-error student**: Student receive an error response after compiling, and students try to compile before going back to the web search.

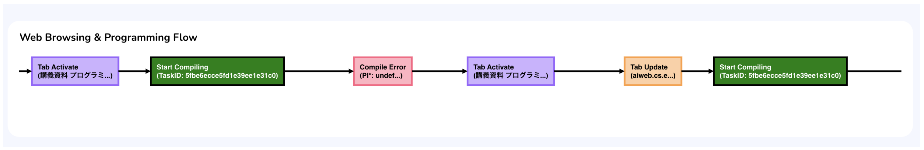

Web Browsing & Programming Flow
Tab Activate (講義資料 プログラミ...)
Start Compiling (TaskID: 5fbe6ecce5fd1e39ee1e31c0)
Compile Error (PI*: undef...)
Tab Activate (講義資料 プログラミ...)
Tab Update (aiweb.cs.e...)
Start Compiling (TaskID: 5fbe6ecce5fd1e39ee1e31c0)

Figure 5: **Try-and-search student:** Student receive an error response after compiling, and students return to the web browsing to find a solution before the following compilation.

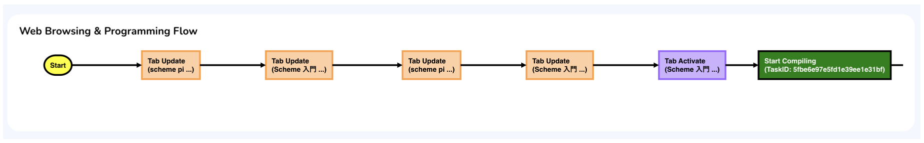

Web Browsing & Programming Flow
Start
Tab Update (scheme pi ...)
Tab Update (Scheme 入門 ...)
Tab Update (scheme pi ...)
Tab Update (Scheme 入門 ...)
Tab Activate (Scheme 入門 ...)
Start Compiling (TaskID: 5fbe6e97e5fd1e39ee1e31bf)

Figure 6: **Cautious student**: Student use web searches before programming. Once the solution is identified, write code from scratch or use a clipboard copy to solve the task.

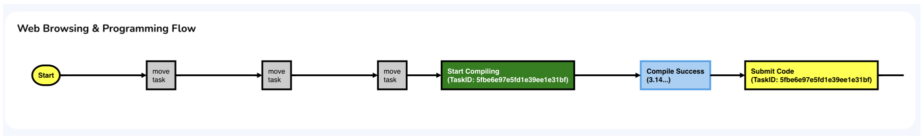

Web Browsing & Programming Flow
Start
move task
move task
move task
Start Compiling (TaskID: 5fbe6e97e5fd1e39ee1e31bf)
Compile Success (3.14...)
Submit Code (TaskID: 5fbe6e97e5fd1e39ee1e31bf)

Figure 7: **Time management student**: Student move to other tasks after starting problem-solving. One participant look for a few questions, whereas the other student look at all questions before starting to solve questions.

5 Result

In this section, we show results obtained from the *TrackThinkDashboard*.

Figure 4 shows a sample student workflow for solving a programming problem using the try-and-error approach. Student compile the code, and after receiving an error response, they modify it and try to recompile it.

Figure 5 shows an example student working on the same task and receiving an error response, but this user decides to go back to the web search before the following compilation. We call this pattern of solving a try-and-search student. The student copies the error message from the compiler and inserts and searches in the web browser. Both students

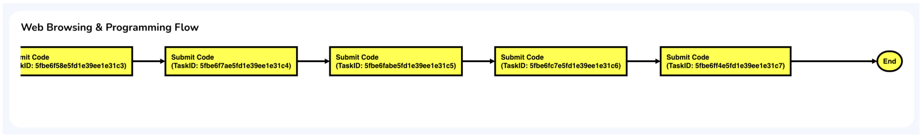
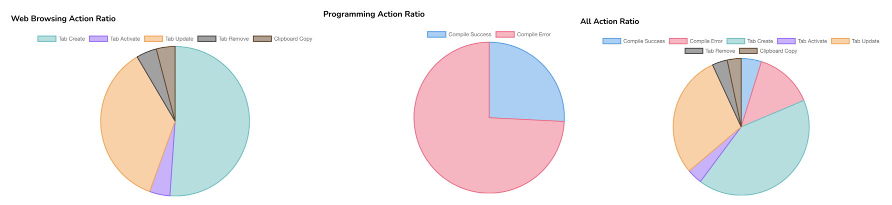
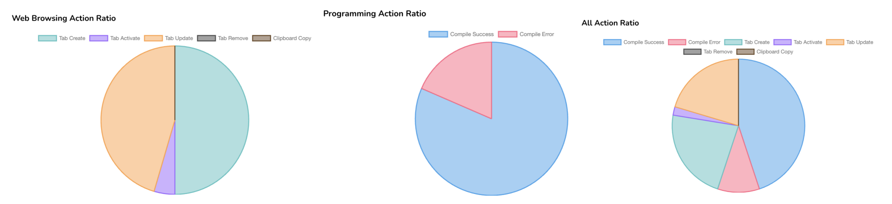

Web Browsing & Programming Flow
mit Code
kID: 5fbe6f58e5fd1e39ee1e31c3)
Submit Code
(TaskID: 5fbe6f7ae5fd1e39ee1e31c4)
Submit Code
(TaskID: 5fbe6fabe5fd1e39ee1e31c5)
Submit Code
(TaskID: 5fbe6fc7e5fd1e39ee1e31c6)
Submit Code
(TaskID: 5fbe6ff4e5fd1e39ee1e31c7)
End

Figure 8: **Double checking student:** Student move to previous questions to double check and submit their code before finishing the experiment.



(a) Sample pie-chart for non-lecture attendee student



(b) Sample pie-chart for lecture attendee student

Figure 9: Pie-chart of selected non-lecture and lecture attendance students. The pie-chart represents a ratio of students' action counts. The left shows an action ratio of web browsing logs, the middle shows the ratio of programming compilation result counts, and the right shows the ratio of all action counts.

receive an error response on the same problem, but each student acts differently to continue solving the problem.

Figure 6 illustrates a sample workflow of a student group tackling a programming task. A defining characteristic of these students is their approach of first gathering information from web pages before attempting to solve the problem. Once they have a clear understanding of the solution, they return to the online IDE to code and submit their work. While two students in this group follow similar workflows, one writes the code from scratch, whereas the other copies sections of code using the clipboard. Both demonstrate a methodical approach, prioritizing understanding over immediate action. We refer to this group as *cautious students*.

Figure 7 presents an example workflow of students who strategically review multiple questions at the start of the experiment. While one student examines only a few questions, another reviews all of them before beginning their solutions. Notably, this behavior was absent among non-lecture students and was exclusive to lecture attendees, who possess domain knowledge of the programming language. These students prioritize solving easier questions first, optimizing their time and effort. We refer to this group as *time management students*.

Figure 8 shows a workflow typical of the *double-checking student*. Before concluding the experiment, student revisit all programming questions to carefully review their submissions. This behavior reflects their thorough and detail-oriented problem-solving approach.

Finally, Figure 9 displays action ratio pie charts from the experiment, comparing non-lecture and lecture attendees. The pie charts illustrate the proportion of time spent on web browsing, programming, and a combination of the two. Students who attended lectures, possessing greater domain knowledge, exhibited a higher success rate in code compilation compared to errors. Conversely, non-lecture students experienced more compilation errors and relied more heavily on web searches. These differences align with findings from prior research [25].

6 Discussion

Our study visualizes a flowchart by combining web browsing and programming data to understand students' self-regulated learning (SRL) workflows. This approach provides insights into how students tackle each task, analyzing their learning activities while uncovering individual differences. Flowcharts capture detailed problem-solving processes, while pie charts highlight frequently used actions, offering a clearer understanding of programming language comprehension [25]. Pie charts help monitor problem-solving progress, while flowcharts enable a closer examination of the steps students take, shedding light on their thought processes and strategies.

Although effective, the current visualization has limitations. One limitation is the lack of action duration representation. Actions are sorted by timestamps, but varying node lengths to reflect duration could help identify when students encounter difficulties. The flowchart could also be enhanced by separating actions into branches, such as web browsing and programming, inspired by Git's branching system [9]. While the current single-line design is simpler, branching could reveal more nuanced workflows in some contexts.

Integrating additional sensor data offers another improvement opportunity. Potential plugins include wearable devices to monitor physiological data for measure stress [7] or fatigue [20], eye-tracking [3] to measure attention [8] or concentration [22], facial recognition technology to assess micro-behaviors or engagement levels [24, 26], and bluetooth based position estimation [2]. Such data could provide deeper insights into students' cognitive states and behaviors.

Our analysis identified five SRL patterns during web browsing and programming: *try and error*, *try and search*, *cautious*, *time management*, and *double checking*. Combining knowledge input (web browsing) with knowledge output (programming) revealed unique problem-solving approaches

[9]`https://git-scm.com/docs/git-branch`

that submitted code alone cannot capture. These insights help teachers understand students better and guide their coaching. Additionally, the dashboard highlights high-performing students' workflows, offering a resource for novice learners. Pie charts also reveal domain knowledge [25].

7 Limitation and Future Work

In this study, several limitations remain. Our study works on visualizing programmers' web browsing and coding behavior in the sequential workflow. We discovered the difference in the student's approach to programming. However, this study lacks direct comparisons between students who used the system and those who did not. In our future study, we will evaluate students' self-regulated learning performance.

Another limitation is the lack of integration of real-time feedback mechanisms and the collection of qualitative data on user experiences. Our work may allow us to visualize the programming and web browsing activities, but we did not implement the system work in real-time; instead, we did post-visualization. Our future work should consider collecting students' SRL abilities over time, and better to receive results in real-time, which then the lecturer can interact with a student for supervising how to improve in programming learning.

The limitation also aligned with the data collection. In this study, we recruited 33 unique (32 males and one female) university students in Japan. The characteristics we observe in the student groups are still a range of Japanese and mainly male. Our future work will be to recruit more participants to discover different types of web browsing and programming behavior.

Other then that, current analysis focuses on individual patterns without considering combinations, such as *cautious* students also using *time management* strategies. Exploring such combinations could provide a better understanding of problem-solving processes. The programming tasks used in this study were relatively simple, focusing on Scheme syntax, leading to limited score variability. Future work should design tasks that encourage diverse strategies and performance levels.

Lastly, overcoming the privacy constraints is another future work. To avoid risk of personal browsing data, we have not made the raw dataset publicly available. However, we are exploring ways to anonymize the data so that the web browsing and coding logs could be shared for reproducibility. We anticipate future releases of partial data under an institutional review framework.

8 Conclusion

In this paper, we introduced *TrackThinkDashboard*, a system that visualizes self-regulated learning (SRL) workflows via flowcharts and pie charts to capture both web browsing and programming activities. By collecting data from 33 university students (with varying backgrounds in Scheme), our dashboard revealed diverse problem-solving patterns that spotlight how students allocate effort between information-seeking and code revisions. Beyond mere identification of learning strategies, these visualizations explicitly support teachers by uncovering when and why certain learners become stuck, and by highlighting frequent error–search cycles or effective workflows from high-performing students. Armed with this data, teachers can provide more targeted guidance—for instance, by reviewing the specific resources a struggling student consults or offering additional examples for recurring error messages. Novice learners can also study the flowcharts of advanced peers to adopt more efficient approaches, such as balancing trial-and-error with timely research on relevant syntax or debugging techniques. Taken together, *TrackThinkDashboard* offers a practical framework for both instructors—who can rapidly identify at-risk students and tailor feedback—and students aiming to refine their metacognitive awareness. Future work will focus on integrating these visualizations into ongoing classroom use, refining the data pipeline for larger-scale or real-time deployments, and evaluating long-term impacts on SRL skill development. We have also addressed manuscript repetition, clarified the paper's scope and audience, and corrected template formatting to better align with publication requirements.

Acknowledgement

This work is partially supported by JSPS KAKENHI Grant Number 23KK0188.

References

[1] Stephen Adams and Chris Hanson. Mit scheme user's manual. *Massachusetts Institute of Technology*, 1995.

[2] Hymalai Bello, Sungho Suh, Bo Zhou, and Paul Lukowicz. Besound: Bluetooth-based position estimation enhancing with cross-modality distillation. In *2024 International Conference on Activity and Behavior Computing (ABC)*, pages 1–10, 2024. doi: 10.1109/ABC61795.2024.10651851.

[3] Ankur Bhatt, Ko Watanabe, Andreas Dengel, and Shoya Ishimaru. Appearance-based gaze estimation with deep neural networks: From data collection to evaluation. *International Journal of Activity and Behavior Computing*, 2024(1):1–15, 2024. doi: 10.60401/ijabc.9.

[4] Joel Brandt, Philip J Guo, Joel Lewenstein, Mira Dontcheva, and Scott R Klemmer. Two studies of opportunistic programming: interleaving web foraging, learning, and writing code. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1589–1598, 2009.

[5] Matthew Carrasco, Eunyee Koh, and Sana Malik. Pophistory: Animated visualization of personal web browsing history. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Hu-*

man Factors in Computing Systems, CHI EA '17, page 2429–2436, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450346566. doi: 10.1145/3027063.3053259. URL `https://doi.org/10.1145/3027063.3053259`.

[6] Helder Da Rocha. *Learn Chart. js: Create interactive visualizations for the web with chart. js 2.* Packt Publishing Ltd, 2019.

[7] Prerna Garg, Jayasankar Santhosh, Andreas Dengel, and Shoya Ishimaru. Stress detection by machine learning and wearable sensors. In *26th International Conference on Intelligent User Interfaces-Companion*, pages 43–45, 2021.

[8] Shoya Ishimaru, Syed Saqib Bukhari, Carina Heisel, Jochen Kuhn, and Andreas Dengel. Towards an intelligent textbook: eye gaze based attention extraction on materials for learning and instruction in physics. In *Proceedings of the 2016 ACM international joint conference on pervasive and ubiquitous computing: Adjunct*, pages 1041–1045, 2016.

[9] Peiling Jiang, Fuling Sun, and Haijun Xia. Log-it: Supporting programming with interactive, contextual, structured, and visual logs. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450394215. doi: 10.1145/3544548.3581403. URL `https://doi.org/10.1145/3544548.3581403`.

[10] Sorin Lerner. Projection boxes: On-the-fly reconfigurable visualization for live programming. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, page 1–7, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450367080. doi: 10.1145/3313831.3376494. URL `https://doi.org/10.1145/3313831.3376494`.

[11] Dastyni Loksa, Lauren Margulieux, Brett A. Becker, Michelle Craig, Paul Denny, Raymond Pettit, and James Prather. Metacognition and self-regulation in programming education: Theories and exemplars of use. *ACM Trans. Comput. Educ.*, 22(4), September 2022. doi: 10.1145/3487050. URL `https://doi.org/10.1145/3487050`.

[12] Jihed Makhlouf, Yutaka Arakawa, and Ko Watanabe. A privacy-aware browser extension to track user search behavior for programming course supplement. In Takahiro Hara and Hirozumi Yamaguchi, editors, *Mobile and Ubiquitous Systems: Computing, Networking and Services*, pages 783–796, Cham, 2022. Springer International Publishing. ISBN 978-3-030-94822-1.

[13] Mark Marron. Log++ logging for a cloud-native world. In *Proceedings of the 14th ACM SIGPLAN International Symposium on Dynamic Languages*, DLS 2018, page 25–36, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450360302. doi: 10.1145/3276945.3276952. URL `https://doi.org/10.1145/3276945.3276952`.

[14] Dan Morris, Meredith Ringel Morris, and Gina Venolia. Searchbar: A search-centric web history for task resumption and information re-finding. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, page 1207–1216, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605580111. doi: 10.1145/1357054.1357242. URL `https://doi.org/10.1145/1357054.1357242`.

[15] Kazuma Nagano, Yutaka Arakawa, and Keiich Yasumoto. Trackthink: A tool for tracking a thought process on web search. In *Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers*, UbiComp '17, page 681–687, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450351904. doi: 10.1145/3123024.3129267. URL `https://doi.org/10.1145/3123024.3129267`.

[16] Phuong T. Nguyen, Juri Di Rocco, Davide Di Ruscio, and Massimiliano Di Penta. Crossrec: Supporting software developers by recommending third-party libraries. *Journal of Systems and Software*, 161:110460, 2020. ISSN 0164-1212. doi: https://doi.org/10.1016/j.jss.2019.110460. URL `https://www.sciencedirect.com/science/article/pii/S0164121219302341`.

[17] Luca Ponzanelli, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Michele Lanza. Prompter: Turning the ide into a self-confident programming assistant. *Empirical Software Engineering*, 21:2190–2231, 2016.

[18] Luca Ponzanelli, Simone Scalabrino, Gabriele Bavota, Andrea Mocci, Rocco Oliveto, Massimiliano Di Penta, and Michele Lanza. Supporting software developers with a holistic recommender system. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 94–105, 2017. doi: 10.1109/ICSE.2017.17.

[19] James Prather, Brett A. Becker, Michelle Craig, Paul Denny, Dastyni Loksa, and Lauren Margulieux. What do we think we think we are doing? metacognition and self-regulation in programming. In *Proceedings of the 2020 ACM Conference on International Computing Education Research*, ICER '20, page 2–13, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370929. doi: 10.1145/3372782.3406263. URL `https://doi.org/10.1145/3372782.3406263`.

[20] Elsen Ronando and Sozo Inoue. Improving fatigue detection with feature engineering on physical activity accelerometer data using large language models. *International Journal of Activity and Behavior Computing*, 2024(2):1–22, 2024. doi: 10.60401/ijabc.18.

[21] Leonardo Silva, António Mendes, Anabela Gomes, and Gabriel Fortes. What learning strategies are used by programming students? a qualitative study grounded on the self-regulation of learn-

ing theory. *ACM Trans. Comput. Educ.*, 24(1), February 2024. doi: 10.1145/3635720. URL `https://doi.org/10.1145/3635720`.

[22] Saki Tanaka, Airi Tsuji, and Kaori Fujinami. Eye-tracking for estimation of concentrating on reading texts. *International Journal of Activity and Behavior Computing*, 2024(1):1–21, 2024. doi: 10.60401/ijabc.10.

[23] Yuta Taniguchi, Tsubasa Minematsu, Fumiya Okubo, and Atsushi Shimada. Visualizing source-code evolution for understanding class-wide programming processes. *Sustainability*, 14(13), 2022. ISSN 2071-1050. doi: 10.3390/su14138084. URL `https://www.mdpi.com/2071-1050/14/13/8084`.

[24] Ko Watanabe, Yusuke Soneda, Yuki Matsuda, Yugo Nakamura, Yutaka Arakawa, Andreas Dengel, and Shoya Ishimaru. Discaas: Micro behavior analysis on discussion by camera as a sensor. *Sensors*, 21 (17):5719, 2021.

[25] Ko Watanabe, Yuki Matsuda, Yugo Nakamura, Yutaka Arakawa, and Shoya Ishimaru. How do programmers use the internet? discovering domain knowledge from browsing and coding behaviors. In *2022 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, pages 605–610. IEEE, 2022.

[26] Ko Watanabe, Tanuja Sathyanarayana, Andreas Dengel, and Shoya Ishimaru. Engauge: Engagement gauge of meeting participants estimated by facial expression and deep neural network. *IEEE Access*, 2023.

[27] Ko Watanabe, Seiya Tanaka, Andrew Vargo, Koichi Kise, and Shoya Ishimaru. Trackthink camera: A tool for tracking facial and body information while web browsing. In *AAGPW@ AIED*, pages 7–14, 2023.

[28] Jeffrey Wong and Jason I. Hong. Making mashups with marmite: Towards end-user programming for the web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, page 1435–1444, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595935939. doi: 10.1145/1240624.1240842. URL `https://doi.org/10.1145/1240624.1240842`.

[29] Luyan Xu, Zeon Trevor Fernando, Xuan Zhou, and Wolfgang Nejdl. Logcanvas: visualizing search history using knowledge graphs. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 1289–1292, 2018.

[30] Barry J Zimmerman and Dale H Schunk. *Self-regulated learning and academic achievement: Theoretical perspectives.* Routledge, 2001.