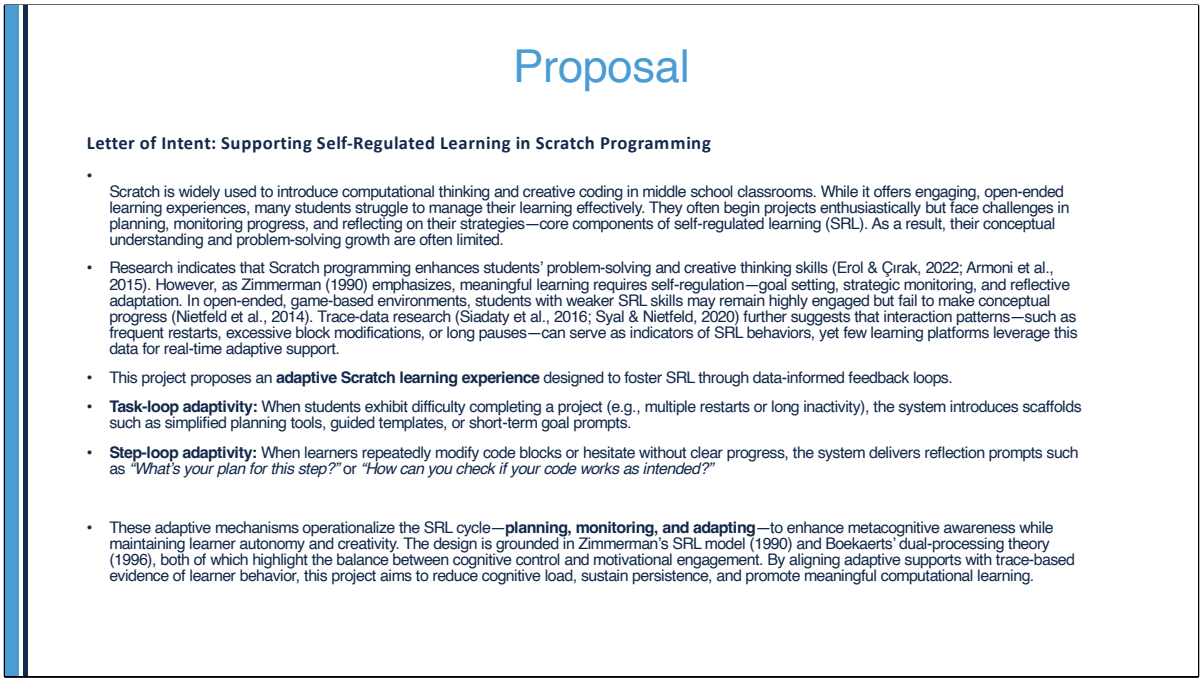
Code to Learn: Adaptive Support for Self-Regulated Leanring in Scratch

(Helping students plan, monitor and reflect during programming tasks using step and task-level adaptivity)

Peizhen Li

EDUC 846: *Personalized & Adaptive Learning*



This project proposes a personalized learning module designed to strengthen students' self-regulated learning (SRL) skills within Scratch, a block-based programming environment widely used in middle school computing education. Although Scratch is highly engaging and accessible, many learners struggle with the metacognitive processes needed to learn effectively — specifically planning, monitoring, and reflection (Zimmerman, 1990). As prior research suggests, students frequently begin programming tasks with enthusiasm but lack the strategic behaviors required to work systematically, debug efficiently, or evaluate their progress (Armoni et al., 2015; Erol & Çırak, 2022). These challenges often result in disorganized code, repeated trial-and-error attempts, and limited transfer of computational concepts. This learning design embeds SRL-aligned adaptivity into Scratch through two feedback layers: the *step loop* and the *task loop*. Step-loop adaptivity targets micro-moments during coding, detecting when students exhibit low planning behaviors (e.g., narrow block exploration), ineffective monitoring (e.g., few test runs or lack of debugging efforts), or signs of hesitation. When such patterns appear, the system triggers brief prompts that guide learners toward more productive strategies, such as "Try running your project after each small change" or "Explore additional block categories to develop your plan." These prompts do not dictate solutions; instead, they support autonomy while encouraging strategic engagement, consistent with

Boekaerts' dual-processing model (1996).
Task-loop adaptivity occurs upon project submission. Here, the system classifies SRL performance by analyzing planning breadth, monitoring intensity, and reflection depth (e.g., success-run rate, note updates). Students then receive a tailored feedback summary describing their SRL strengths and areas for improvement. Such holistic feedback supports metacognitive awareness and long-term skill development. Grounded in SRL theory and informed by trace-data analytics (Siadaty et al., 2016; Syal & Nietfeld, 2020), this project aims to enhance computational thinking while helping students become more reflective, strategic, and self-directed learners in creative coding environments.

Logic Model

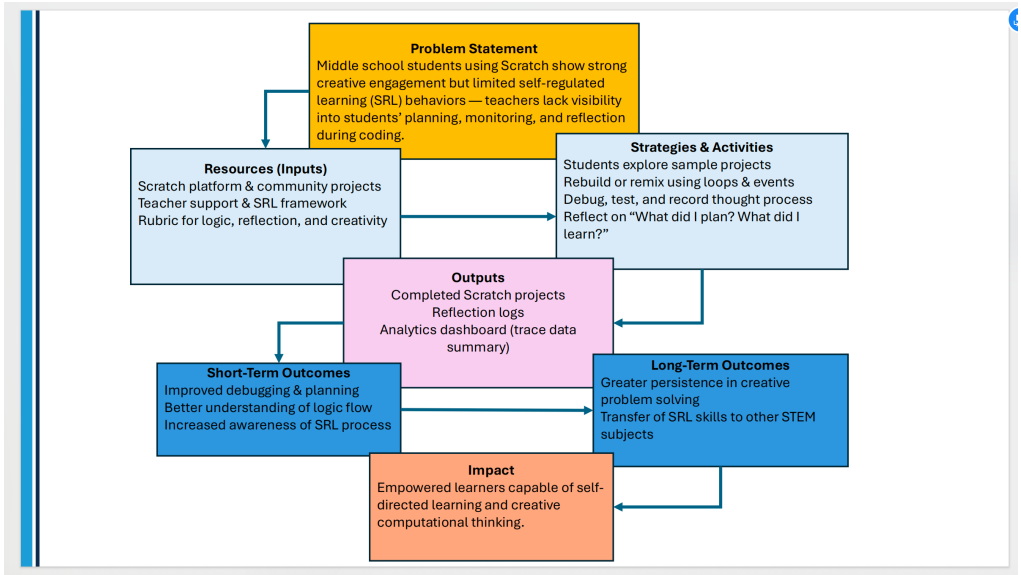
Problem Statement Lack od SRL support and visibility in Scratch-based coding environments for middle school students				
Theoretical framework & evidence that suggests promise of a PL-based solution				
Input(s)	**Task Design Details & Output of the PL Enactments**	**Immediate Outcome(s)**	**Long-term outcome(s)**	**Impacts**
Middle schools students, teachers, Scratch platform,SRL model, analytics tool.	Coding tasks with adaptive prompts, reflection questions,system monitoring of trace data Students SRL behavior data, reflective responses, feedback-summary for teachers.	Increased students SRL awareness (planning, monitoring , reflection) Improved persistence, problem-solving skills, enagement	Strenthened computational thinking and self-directed learng skills	Greater readiness for advanced STEM learning and problem solving skills

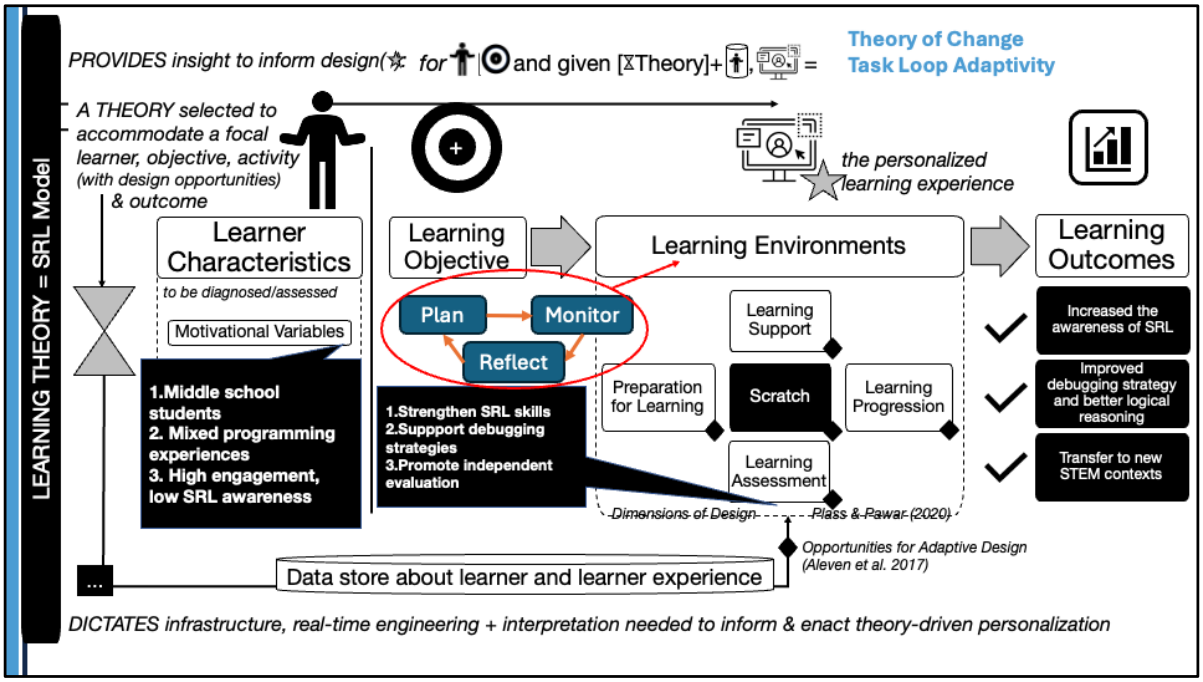
The logic model provides a structural overview of the conditions, processes, and mechanisms guiding this personalized learning design. It articulates how learner characteristics, instructional objectives, learning supports, and assessment strategies interact within an SRL-focused framework. Middle school learners often approach Scratch with enthusiasm but inconsistent strategic behaviors; therefore, the model positions SRL as both the instructional target and the personalization driver. Planning, monitoring, and reflection are mapped onto measurable indicators extracted from Scratch interactions, such as category diversity, run counts, debug attempts, and note-update behaviors.

The model establishes adaptive opportunities by aligning SRL theory with actionable trace data. In planning, learners' use of block categories serves as an indicator of idea breadth; in monitoring, run counts and debugging behaviors reflect testing frequency and strategy use; and in reflection, success-run rate and documentation behaviors capture students' evaluative reasoning. These indicators feed into adaptive mechanisms that shape both step-loop and task-loop feedback, ensuring instructional alignment with learners' moment-to-moment behaviors and overall task completion patterns.

By integrating learner data with SRL-centered instructional design, the logic model supports meaningful personalization that enhances cognitive engagement, debugging

competency, and metacognitive awareness. Ultimately, this model frames the technical and pedagogical infrastructure required to activate theory-driven adaptivity in Scratch.


Problem Statement
Middle school students using Scratch show strong creative engagement but limited self-regulated learning (SRL) behaviors — teachers lack visibility into students' planning, monitoring, and reflection during coding.
Resources (Inputs)
Scratch platform & community projects
Teacher support & SRL framework
Rubric for logic, reflection, and creativity
Strategies & Activities
Students explore sample projects
Rebuild or remix using loops & events
Debug, test, and record thought process
Reflect on "What did I plan? What did I learn?"
Outputs
Completed Scratch projects
Reflection logs
Analytics dashboard (trace data summary)
Short-Term Outcomes
Improved debugging & planning
Better understanding of logic flow
Increased awareness of SRL process
Long-Term Outcomes
Greater persistence in creative problem solving
Transfer of SRL skills to other STEM subjects
Impact
Empowered learners capable of self-directed learning and creative computational thinking.



In this revised extension model, I integrated Bernacki's personalized learning framework with Zimmerman's SRL cycle. The outer structure remains the same—from learner characteristics, to objectives, to learning environments and outcomes—but inside, I explicitly embedded the SRL engine: **planning, monitoring, and reflecting**.

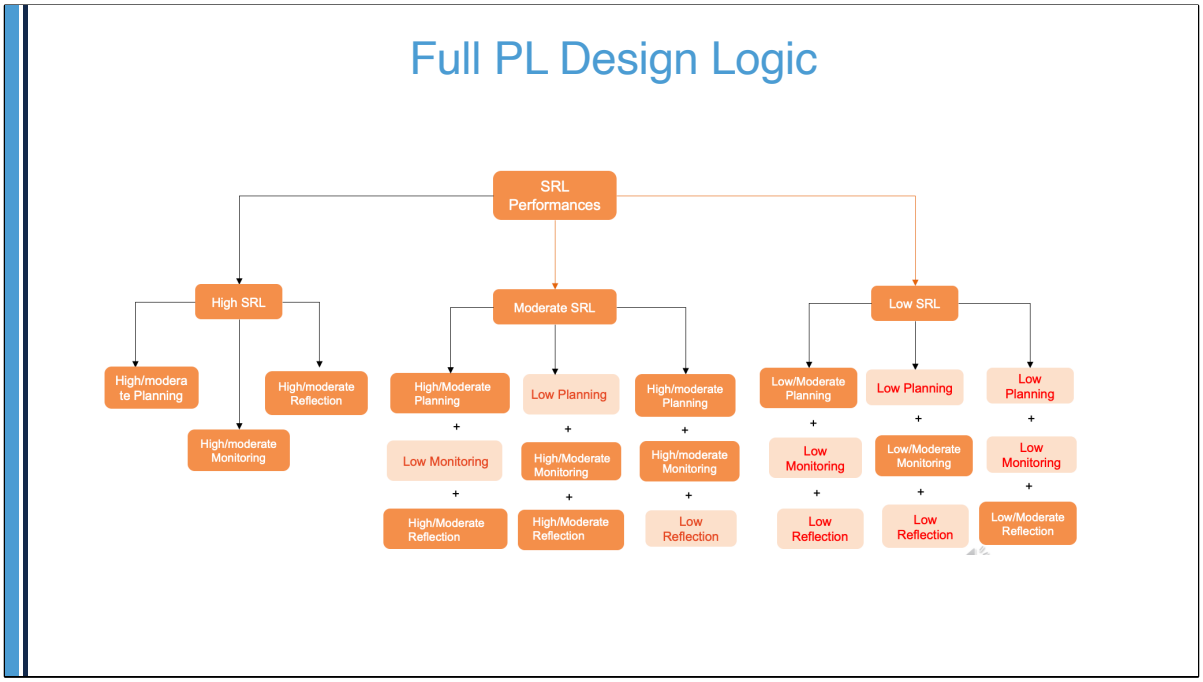
This SRL loop drives all adaptive decisions in my design. As students work in Scratch, their behaviors in these three phases trigger tailored Step-Loop and Task-Loop supports. This allows the system to respond in real time, strengthen SRL strategies, and ultimately improve debugging skills, logical reasoning, and students' awareness of their own learning processes. The adaptive Scratch module is intended to produce meaningful improvements in students' self-regulated learning (SRL). The theory of change begins with the assumption that Scratch offers an open-ended, creative environment, but one in which students frequently struggle to manage their learning processes—especially planning, monitoring, and reflecting. These SRL components are essential to successful computational thinking, yet they are rarely scaffolded explicitly within introductory programming experiences. The PL model addresses this gap by embedding adaptive feedback loops that guide students' learning processes without restricting autonomy.

Because SRL manifests through visible interactions—such as breadth of block use,

frequency of test runs, debugging attempts, and note updates—these actions can be captured as trace data and mapped to key SRL phases. The Step Loop leverages these micro behaviors during the task to provide just-in-time scaffolds. For example, if the learner has spent time building but used only one or two block categories, the system interprets this as insufficient planning and delivers a gentle prompt encouraging broader exploration. Similarly, low monitoring behaviors—such as infrequent test runs or lack of debugging attempts—trigger strategic hints that help students evaluate and revise their approach.

The Task Loop forms the second adaptive layer by synthesizing all observable behaviors into a final SRL profile after the project is submitted. This macro-level feedback reinforces metacognitive awareness. By summarizing strengths (e.g., strong monitoring) and areas for growth (e.g., limited reflection), the system supports learners in understanding *how* they approached the task, not just whether their code "worked." This aligns with the extended PL framework's emphasis on personalization that extends beyond content mastery into cognitive and metacognitive development.

Ultimately, the theory of change proposes that these two layers of adaptivity—moment-to-moment scaffolding and end-of-task reflection—work together to strengthen SRL habits over time. As learners internalize these strategies, they become more autonomous, strategic, and reflective programmers. The expected long-term impact is a more confident learner who can independently plan, test, debug, and evaluate their work across a variety of computational tasks.



This slide presents the complete Personalized Learning (PL) design logic that structures the adaptive experience in this Scratch-based SRL module. The purpose of this logic is to show the coherent flow from data inputs, to SRL detection, to adaptive feedback delivery. The model integrates both micro-level adaptivity (Step Loop) and macro-level adaptivity (Task Loop), aligning with the Extended PL Model and supporting students' development of planning, monitoring, and reflection skills during programming tasks.

The logic begins with **traceable behavioral indicators** collected as learners interact with Scratch. These include block-category diversity, block-type counts, run frequency, tutorial time, debug-tip clicks, success-run rate, and note-update behaviors. Each of these behaviors corresponds directly to an SRL phase: planning indicators reflect the breadth and intentionality of students' initial ideas; monitoring indicators capture iterative testing and debugging strategies; reflection indicators reveal how students evaluate outcomes and articulate reasoning.

Together, the Step Loop and Task Loop form an integrated adaptive system grounded in SRL theory and personalized learning design. This complete logic ensures that learners receive meaningful, context-sensitive support during and after task engagement, strengthening both computational and metacognitive skills over time.

Planning phase:

Based on the current Scatch platform, I analyze the total categories and block types within each catogories, then create the threshold regarding to Low, Medium and High of planning phase

	Block Categories (Total count 8)	**Block Types** (based on block category)
Low	0~2	0~6
Medium	3~6	6~18
High	7~8	> 20

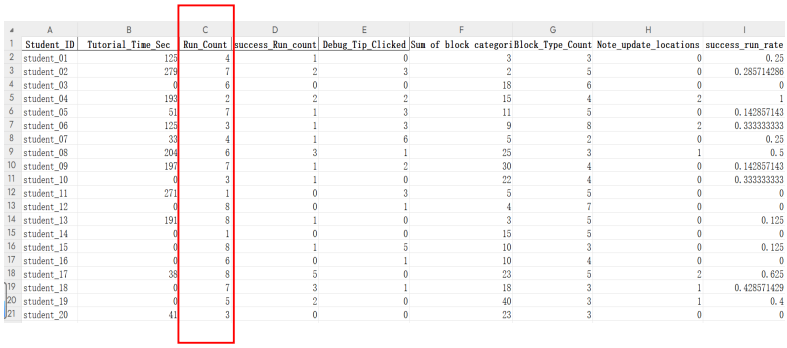
•**High planning** → students use **7–8 block categories** (i.e., they explore most of Scratch's available function areas: motion, looks, control, sound, sensing, events, operators, variables).
•**Moderate planning** → they use 3–6 categories.
•**Low planning** → they use 0–2 categories, usually repeating the same block types.

The Planning Phase represents the first stage of self-regulated learning (SRL) within the Scratch environment, and it serves as a critical component of this personalized learning design. In computational tasks such as the "Next Dog" project, planning is visible through the ways students explore ideas, organize actions, and decide which blocks to use before fully executing their program. This slide outlines the specific indicators, thresholds, and adaptive triggers used to detect planning behaviors and scaffold students' early decision-making.
In this design, planning is assessed primarily through two observable behaviors: **block-category breadth** and **block-type count**. Block-category breadth reflects the diversity of conceptual tools learners draw upon; categories such as Motion, Looks, Sound, Control, and Events represent different dimensions of computational thinking. Students who use only one or two categories early in the task typically show narrow planning and limited exploration of design possibilities. Therefore, block-category breadth functions as the *primary* indicator for planning level: 0–2 categories indicate Low Planning, 3–6 indicate Medium Planning, and 7–8 indicate High Planning.
Block-type count serves as a *secondary* indicator used to refine the planning classification. This metric captures the number of unique blocks a learner places during the initial planning window. Fewer than six blocks often reflects superficial or incomplete planning, whereas a moderate number of blocks suggests early

structuring of ideas. If a student uses multiple categories but very few total blocks, the system interprets this mismatch as shallow planning and may classify the learner at a lower level. Conversely, if a student uses several categories and a high number of blocks, their planning level can be upgraded.

Monitoring phase:

To establish conceptual boundaries for monitoring levels, I generated a small simulated dataset (n=20) representing typical Scratch user behavior. Most students executed their project between 3–7 times, while those with fewer than three runs tended to complete tasks without iterative testing. Therefore, I classified fewer than three runs as low monitoring, 3–6 as moderate, and more than six as high monitoring.



	A	B	C	D	E	F	G	H	I
1	Student_ID	Tutorial_Time_Sec	Run_Count	success_Run_count	Debug_Tip_Clicked	Sum of block categori	Block_Type_Count	Note_update_locations	success_run_rate
2	student_01	125	4	1	0	3	3	0	0.25
3	student_02	279	7	2	3	2	5	0	0.285714286
4	student_03	0	6	0	0	18	6	0	0
5	student_04	193	2	2	2	15	4	2	1
6	student_05	51	7	1	3	11	5	0	0.142857143
7	student_06	125	3	1	3	9	8	2	0.333333333
8	student_07	33	4	1	6	5	2	0	0.25
9	student_08	204	6	3	1	25	3	1	0.5
10	student_09	197	7	1	2	30	4	0	0.142857143
11	student_10	0	3	1	0	22	4	0	0.333333333
12	student_11	271	1	0	3	5	5	0	0
13	student_12	0	8	0	1	4	7	0	0
14	student_13	191	8	1	0	3	5	0	0.125
15	student_14	0	1	0	0	15	5	0	0
16	student_15	0	8	1	5	10	3	0	0.125
17	student_16	0	6	0	1	10	4	0	0
18	student_17	38	8	5	0	23	5	2	0.625
19	student_18	0	7	3	1	18	3	1	0.428571429
20	student_19	0	5	2	0	40	3	1	0.4
21	student_20	41	3	0	0	23	3	0	0

The Monitoring Phase represents the second major component of self-regulated learning (SRL), and in a Scratch programming environment, it captures how learners track progress, test their ideas, and make informed adjustments while working on an open-ended task. Monitoring is observable in students' iterative behaviors—specifically their test runs, debugging actions, and use of tutorials or helpful resources. These indicators reveal whether learners are evaluating their work strategically or relying on passive or guess-based approaches. This slide outlines how monitoring is operationalized and how adaptive supports are triggered during task engagement.

Monitoring is assessed using three primary behavioral indicators: **run count**, **debug-tip clicks**, and **tutorial engagement time**. Among these, run count functions as the *primary* monitoring indicator. Frequent testing is a hallmark of strong monitoring, as it reflects continuous checking of progress and iterative refinement. A run count below three during the early or middle stages of building is classified as Low Monitoring; such minimal testing suggests the learner is not evaluating their work systematically. Higher run frequencies indicate more strategic monitoring.

Debug-tip clicks and tutorial time serve as *secondary* indicators that refine the monitoring classification. Learners who interact with debugging tools or spend meaningful time viewing tutorials demonstrate intentional efforts to solve problems

and deepen understanding. When tutorial engagement exceeds approximately 120 seconds or debugging interactions surpass six clicks, monitoring is classified as High. Moderate tutorial time (30–120 seconds) or moderate debugging activity (4–6 clicks) contributes to a Medium Monitoring classification. Conversely, when run count is high but debug/tutorial engagement is minimal, the system may still identify gaps in monitoring depth.

Reflection phase:

Success run rate (Success run count/ total run count)	
0~0.3	Low
0.31~0.6	Medium
0.61~1	High

Effectiveness and quality of understanding
Thresholds for success-run rate were derived from simulated learner data (n=20) to represent low, moderate, and high reflection outcomes. Although the dataset is illustrative, these values mirror realistic behavioral differences observed in coding-based learning environments

	A	B	C	D	E	F	G	H	I
1	Student_ID	Tutorial_Time_Sec	Run_Count	success_Run_count	Debug_Tip_Clicked	Sum of block categori	Block_Type_Count	Note_update_locations	success_run_rate
2	student_01	125	4	1	0	3	3	0	0.25
3	student_02	279	7	2	3	2	5	0	0.285714286
4	student_03	0	6	0	0	18	6	0	0
5	student_04	193	2	2	2	15	4	2	1
6	student_05	51	7	1	3	11	5	0	0.142857143
7	student_06	125	3	1	3	9	8	2	0.333333333
8	student_07	33	4	1	6	5	2	0	0.25
9	student_08	204	6	3	1	25	3	1	0.5
10	student_09	197	7	1	2	30	4	0	0.142857143
11	student_10	0	3	1	0	22	4	0	0.333333333
12	student_11	271	1	0	3	5	5	0	0
13	student_12	0	8	0	1	4	7	0	0
14	student_13	191	8	1	0	3	5	0	0.125
15	student_14	0	1	0	0	15	5	0	0
16	student_15	0	8	1	5	10	3	0	0.125
17	student_16	0	6	0	1	10	4	0	0
18	student_17	38	8	5	0	23	5	2	0.625
19	student_18	0	7	3	1	18	3	1	0.428571429
20	student_19	0	5	2	0	40	3	1	0.4
21	student_20	41	3	0	0	23	3	0	0

The Reflection Phase represents the final and most metacognitive component of self-regulated learning (SRL). In the context of Scratch programming, reflection occurs after learners observe their project's outcome, evaluate its functionality, and articulate what worked, what did not, and why. This phase is often the least visible in open-ended tasks because students may run their code repeatedly but rarely document their reasoning or think explicitly about the effectiveness of their strategies. Consequently, this design emphasizes two measurable indicators of reflection: **success-run rate** and **note-update behaviors**.

Step loop feedback based on Planning

Detected Behavior	Phase	Hypothetical Step Feedback
Student adds **<3** block categories	Planning (Low)	Try using more block types (e.g., motion, looks) to make your animation richer.
Student adds **3~6** block categories	Planning (Moderate)	Great start! Before running, consider checking whether you've set your character's starting position or added necessary control blocks.
Student adds **7~8** block categories	Planning (High)	You've planned your structure very well — nice use of multiple block types. When ready, try your first test run to see how everything works together.

Step loop feedback based on Monitoring

Detected Behavior	Phase	Hypothetical Step Feedback
Student runs **< 3** times of the project	Monitoring(Low)	You've tested only once — try running again after small code changes to debug more effectively.
Student runs the project **3~6** times	Monitoring(Moderate)	Nice debugging so far. If something still looks off, try using the debug tips or checking your tutorials for similar examples.
Student runs **> 6** times of the project	Monitoring(High)	Strong monitoring! You're testing and revising iteratively — keep paying attention to how each change affects your dog's movement.

Step loop feedback based on Reflection

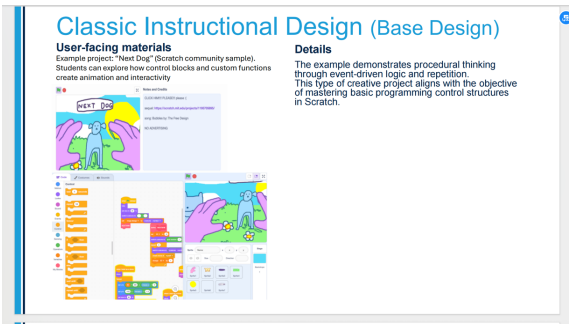
Detected Behavior	Phase	Hypothetical Step Feedback
Student never edits notes	Reflection (Low)	Before submitting, describe what worked or didn't in your notes — it helps your future self understand your process.
Student edits notes **1 ~ 2**	Reflection (Moderate)	You've improved your project — try writing a short note about what change made the biggest difference.
Student edits notes **> 3** times	Reflection (High)	Great reflection! Your notes show thoughtful adjustments. Consider summarizing your main learning takeaway for future projects.

Task loop feedback based on overall SRL behaviours

Metric	Feedback
Final SRL **(Low)**	You completed the project — well done. Your work suggests limited planning and few test runs during development. Next time, try exploring (expected number of) block categories and testing your code more frequently — these steps will help you improve faster
Final SRL **(Moderate)**	Great job on the animation. You engaged in multiple test cycles and made several improvements. To reach a higher SRL level, consider adding (expected number of) reflective notes and exploring more advanced block combinations.
Final SRL **(High)**	Excellent work — your project shows strong planning, iterative testing, and thoughtful reflection. You're ready for more challenging tasks that encourage creativity and deeper debugging.

Classic Instructional Design

This module uses *'The Next Dog'* — a community-created Scratch project — as an anchor example. It represents a manageable yet conceptually rich task that requires learners to plan a sequence of code blocks, monitor the project's movement and interactions, and reflect on outcomes through debugging. Choosing an existing community project provides an authentic and replicable scenario while allowing the module to focus on scaffolding SRL processes rather than programming complexity.



1.Student Task: Create *The Next Dog* animation.
2.System Detection Logic: Categorizes SRL behaviors (based on planning, monitoring, reflection).
3.Feedback Loop:

1. *Step Loop (hypothetical)*: Immediate micro-prompts based on in-progress patterns (conceptual examples only).
2. *Task Loop (implemented)*: Summative SRL feedback upon submission (e.g., dashboard summary, reflection tips).

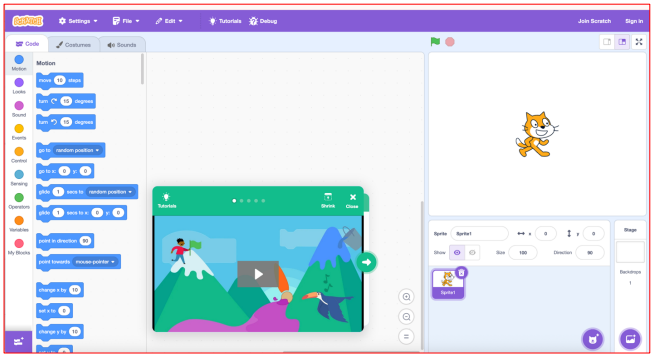
This slide repositions the project within the framework of **classic instructional design**, grounding the personalized learning elements in a clear instructional foundation. While the extended personalized learning model emphasizes adaptivity and data-driven feedback, classic instructional design focuses on structuring learning goals, instructional tasks, materials, and assessment activities in ways that support all learners, regardless of their SRL profile. In this Scratch project, classic instructional design articulates the core learning experience before any adaptive prompts or SRL-driven logic are applied.
The instructional goal is to help learners create a functioning animation using Scratch—in this case, "The Next Dog." The task is intentionally open-ended yet structured enough to give students a clear entry point into computational thinking.

The first stage is the **Step Loop**, which operates during the construction of the Scratch project. The Step Loop is triggered by early signals of low SRL—such as narrow use of block categories or very few test runs. Its prompts are short and actionable, designed to nudge students toward stronger strategies without interrupting autonomy. For example, if the learner has spent thirty seconds building but used only one block category, the system shows a light scaffold encouraging exploration of new block types. Similarly, infrequent testing prompts a reminder to run the project more often.

These micro adaptive supports help learners regulate in real time.
The second stage is the **Task Loop**, which activates after the project is submitted. The system synthesizes planning, monitoring, and reflection indicators into a final SRL classification (Low, Medium, or High). It then delivers personalized feedback summarizing the learner's SRL strengths and targeted growth suggestions. This feedback focuses on process—strategies the learner used or neglected—rather than on code correctness alone. The Task Loop reinforces metacognitive awareness by helping students understand *how* they approached the task and what they might adjust in future projects.

Classic Instructional Design (Base Design)

User-facing instructions/objectives page



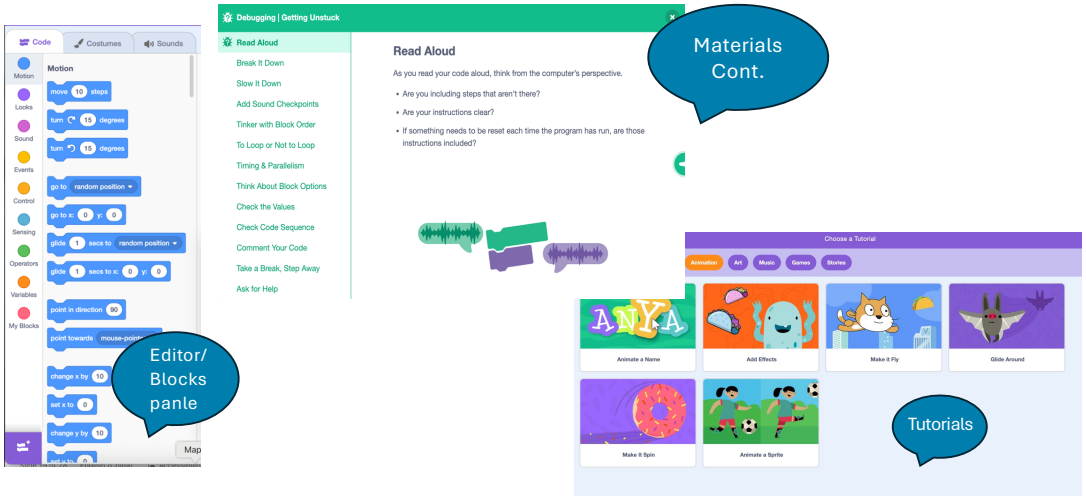
Learning objectives:

- Students will design an animation using Scratch
- Students will follow structured steps
- Students will test and debug their code

Classic instructional design first defines the **learning objectives**, such as: understanding basic Scratch block categories, constructing a sequence of actions, and evaluating project functionality through testing. These objectives ensure that learners know what success looks like and that the task aligns with developmentally appropriate expectations for middle school students.

Classic Instructional Design (Base Design)

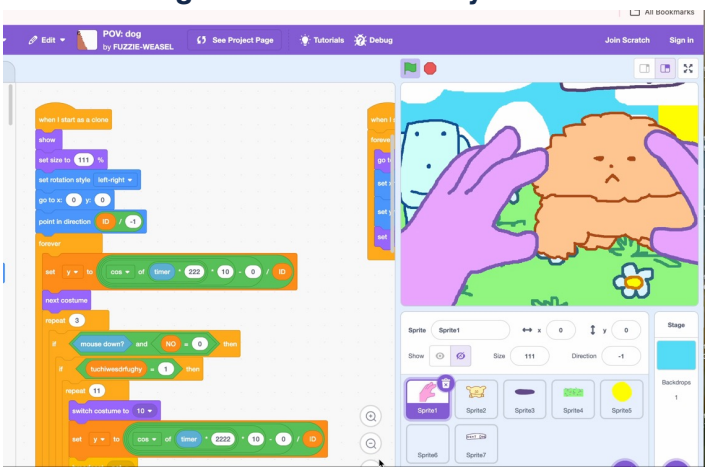
User-facing materials



Next, classic instructional design specifies the **materials and learning supports** available to all students. These include Scratch's familiar drag-and-drop interface, block palettes, sprite editor, example projects, tutorials, and reference resources. Importantly, these materials serve as the universal design layer—ensuring the task is accessible before introducing personalization. Students are provided with clear instructions, sample configurations, and opportunities to explore block functions independently.

Classic Instructional Design (Base Design)

User-facing assessment activity

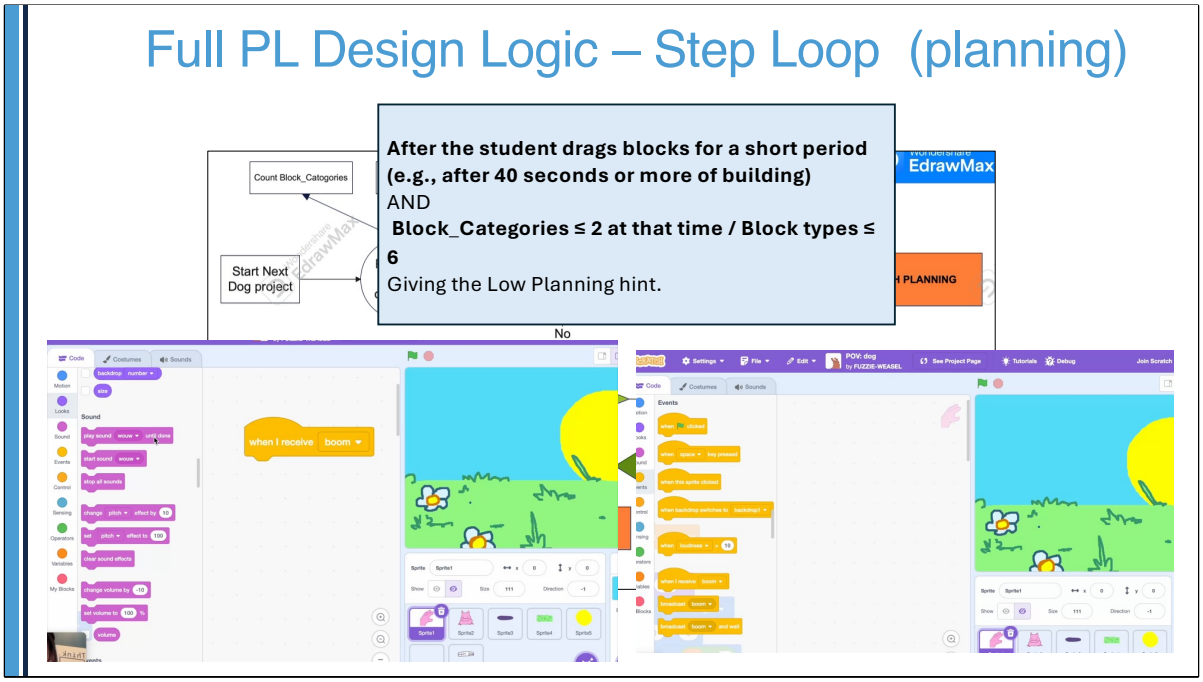


Rubrics:
- When user clicks 'next dog' button, the viewpoint changes to the next dog
- Dog responds to user click
- Code runs without errors

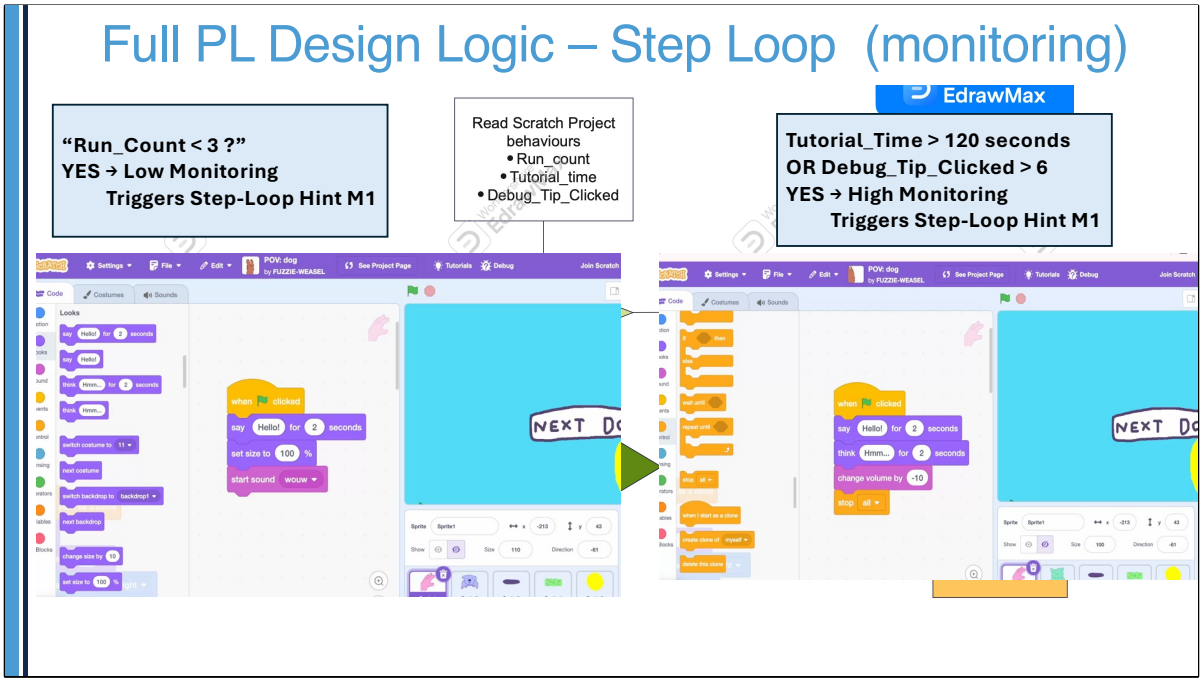
Finally, the **assessment activity** in classic instructional design focuses on evaluating the completed Scratch project based on functionality, clarity of actions, and basic correctness. This includes checking whether sprites move as intended, whether the program runs smoothly, and whether the learner demonstrates basic mastery of the block-based interface.

Only after this classic instructional foundation is established do personalized learning components—namely the Step Loop and Task Loop—enhance the experience by tailoring feedback and scaffolding SRL. This ensures the personalization layer is purposeful rather than arbitrary, supporting both instructional integrity and learner autonomy.

Full PL Design Logic – Step Loop (planning)



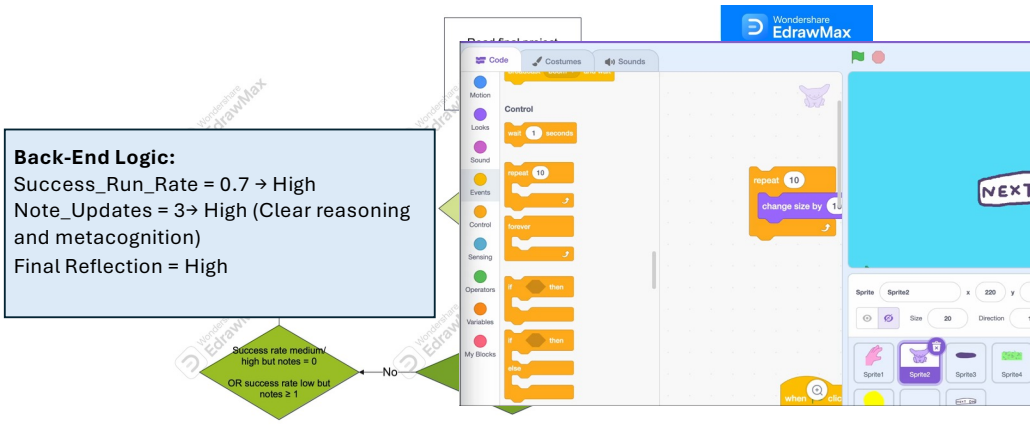
The Step Loop activates during early project building. After roughly 20–30 seconds of construction, the system checks planning indicators. If the learner has used fewer than three categories or fewer than six total blocks, a low-planning hint functions as a just-in-time scaffold. These prompts encourage exploration without dictating solutions. For example: “Try exploring additional block types such as Looks or Sound to expand your plan.” These adaptive nudges aim to broaden students’ planning strategies before they invest significant time in coding.



The Step Loop triggers real-time scaffolds when monitoring is weak. After the system detects that the learner has performed only one or two test runs within a reasonable timeframe, it generates a prompt such as: "Try running your project again after each small change to check your progress." Similarly, if the learner has not used debugging tools, a prompt might encourage them to utilize available supports. These nudges aim to reinforce productive habits without interrupting autonomy.

By embedding these monitoring indicators and adaptive prompts, the system guides students toward more effective evaluation strategies. Strengthening monitoring behaviors supports more efficient debugging, reduces frustration, and fosters SRL skills that extend beyond this specific Scratch task.

Full PL Design Logic – Step Loop (reflection) / post task only

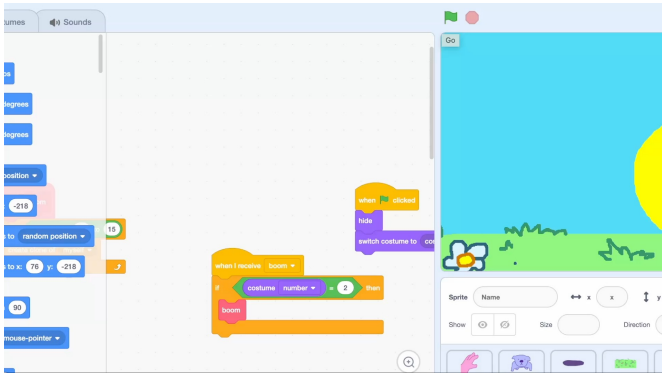


Success-run rate captures how often the project executes as intended relative to total test runs. Although Scratch does not provide built-in success/failure logs, the concept is simulated here to represent outcome evaluation. A low success-run rate (≤0.3) suggests the student has encountered repeated issues without effectively resolving them. By contrast, a higher success-run rate indicates the learner has iteratively refined and improved their program, demonstrating evaluative reasoning consistent with high reflection.

Note-update behaviors serve as a second, equally important reflection indicator. Scratch's "Notes and Credits" or editor notes allow students to document what changes they made, why they made them, and what they discovered during testing. Zero note updates typically reflect minimal metacognitive engagement. Students may fix issues but fail to articulate their reasoning, which limits learning transfer. One or two brief notes indicate moderate reflection, while three or more detailed notes demonstrate strong evaluative thinking, deeper analysis, and awareness of strategy use.

Full PL Design Logic – Task Loop

- **Success-run rate → Low/Med/High Reflection**
- **+**
- **Note updates**
- **Final SRL (Low/Med/High)**
- **Final feedback message**



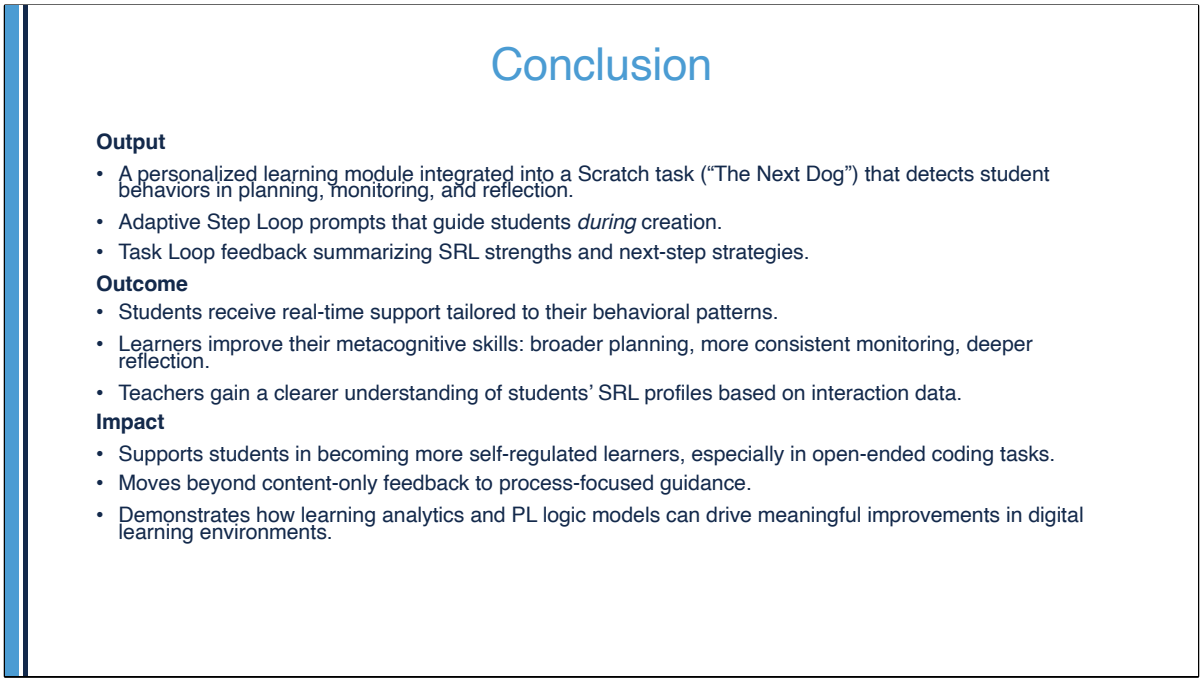
The Task Loop represents the second major adaptive layer in this personalized learning design. Whereas the Step Loop provides just-in-time scaffolding during the construction of the Scratch project, the Task Loop activates after the learner submits the final version of their animation. Its purpose is to synthesize the student's behavioral data across the three SRL phases—planning, monitoring, and reflection—into a coherent profile and provide targeted feedback that supports metacognitive growth. This slide details the logic behind that process and illustrates how learners receive structured, meaningful guidance at the end of the task.

The Task Loop begins by compiling the SRL indicators collected throughout the learning session. Planning is characterized by block-category breadth and block-type counts; monitoring is defined by run frequency, debugging actions, and tutorial engagement; and reflection is captured through success-run rate and note-update behaviors. Each indicator is classified into Low, Medium, or High based on empirically grounded thresholds and refined decision rules. These classifications are then combined to determine an overall SRL level for the completed task.

The system uses a simple, interpretable rule to compute the final SRL profile: if the majority of SRL phases are classified as Low, the learner receives a Low SRL

designation; if a balance of Medium and High indicators is present, the learner is classified as Medium; and if most indicators show strong engagement, the learner receives a High SRL designation. This approach avoids combinatorial complexity and ensures clarity for both learners and educators.

Once the SRL level is determined, the Task Loop generates personalized end-of-task feedback. This feedback summarizes the learner's SRL strengths and highlights concrete strategies for improvement. For instance, a Low SRL feedback message may acknowledge the learner's completion of the task but encourage broader block exploration and more frequent testing. A Medium SRL message might reinforce effective debugging while prompting deeper documentation of reasoning. A High SRL message recognizes strong planning, monitoring, and reflection, and suggests more challenging extensions.



This project demonstrates how a creative coding task—"The Next Dog"—can serve not only as a platform for computational thinking but also as an opportunity to cultivate meaningful self-regulated learning behaviors. By embedding adaptive mechanisms into the task, the module supports learners in becoming more strategic, reflective, and autonomous programmers.

The **outputs** of the project include a fully articulated logic model that connects traceable Scratch behaviors to the SRL constructs of planning, monitoring, and reflection. The project also delivers a detailed set of adaptive rules for the Step Loop and Task Loop, as well as a demonstration of how these rules translate into instructional feedback moments. Through multimodal artifacts—logic diagrams, video demonstrations, and example prompts—the design provides a clear illustration of how personalization can be implemented at both the micro and macro levels of the learning process.

The **outcomes** of this design center on learner experience. As students engage with the task, they receive context-appropriate scaffolds that encourage broader exploration, more consistent monitoring, and richer reflective reasoning. These personalized supports help students avoid common pitfalls in early programming, such as trial-and-error guessing or narrow planning, and instead guide them toward intentional, strategic behaviors that improve both performance and conceptual

understanding. Additionally, the Task Loop's post-submission feedback fosters metacognitive awareness by helping learners interpret their performance and internalize strategies that can be transferred to future tasks.

The broader **impact** of this model extends beyond a single Scratch assignment. By demonstrating how trace data can be transformed into actionable SRL insights, this project illustrates a scalable approach to process-oriented personalization in digital learning environments. The framework can be adapted to other creative coding tasks, interdisciplinary maker activities, and even academic domains that depend on iterative practice. Ultimately, this project reinforces the value of integrating learning analytics, SRL theory, and instructional design to create environments that not only teach content but also build lifelong learning skills.

References

- Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From scratch to "real" programming. *ACM Transactions on Computing Education (TOCE)*, *14*(4), 1-15.
- Boekaerts, M., & Corno, L. (2005). Self-Regulation in the Classroom: A Perspective on Assessment and Intervention. *Applied Psychology*, *54*(2), 199–231. https://doi.org/10.1111/j.1464-0597.2005.00205.x
- Erol, O., & Çırak, N. S. (2022). The effect of a programming tool scratch on the problem-solving skills of middle school students. *Education and Information Technologies*, *27*(3), 4065–4086. https://doi.org/10.1007/s10639-021-10776-w
- Nietfeld, John L, Lucy R Shores, and Kristin F Hoffmann. "Self-Regulation and Gender Within a Game-Based Learning Environment." *Journal of educational psychology* 106.4 (2014): 961–973. Web.
- Siadaty, M., Gasevic, D., & Hatala, M. (2016). Trace-based micro-analytic measurement of self-regulated learning processes. *Journal of Learning Analytics*, *3*(1), 183-214.
- Syal, Samira, and John L Nietfeld. "The Impact of Trace Data and Motivational Self-Reports in a Game-Based Learning Environment." *Computers and education* 157 (2020): 103978-. Web.
- Zimmerman, B. J. (1990). Self-Regulated Learning and Academic Achievement: An Overview. *Educational Psychologist*, *25*(1), 3–17. https://doi.org/10.1207/s15326985ep2501_2