

# Intelligent Support for Collaborative Learning in Advanced Computer Science Courses — From Worked Examples to Data-Driven Insights

**Sreecharan Sankaranarayanan**

CMU-LTI-25-004

Mar 07, 2025

Language Technologies Institute  
School of Computer Science  
Carnegie Mellon University  
Pittsburgh, PA 15213

## Thesis Committee:

**Carolyn Penstein Rosé**, Language Technologies Institute and Human-Computer Interaction Institute, Carnegie Mellon University (Chair)

**Alan W. Black**, Language Technologies Institute, Carnegie Mellon University

**Graham Neubig**, Language Technologies Institute, Carnegie Mellon University

**Candace Thille**, Graduate School of Education, Stanford University

*Submitted in partial fulfillment of the requirements  
for the degree of Doctor of Philosophy in Language and Information Technology.*

Copyright © 2025 Sreecharan Sankaranarayanan

May 3, 2025  
DRAFT

**Keywords:** conversational agents, artificial intelligence in education, reinforcement learning, computer-supported collaborative learning, worked examples, computer science education, higher education, vocational training

*For my parents, who had my back when I didn't have my own, and my sister, who will always inspire me to be the version of myself that she sees.*



## Abstract

In this dissertation, we challenge the preeminence of problem-solving practice in the pedagogy of advanced computer science courses by comparing it to worked examples. In doing so, we not only extend the theory of example-based learning to this novel context, but make significant technological contributions to the infrastructure for conducting AI-facilitated educational experiments at scale. To conduct these experiments, we integrate the Bazaar AI Conversational Agent framework into two courses on the Sail() platform, thus enabling the rapid deployment of AI-facilitated interventions and continuous data collection to support their experimental evaluation.

The findings reveal, in line with theory, that maximizing time spent with worked examples improves conceptual learning. Some problem-solving practice remains necessary for implementation proficiency, especially when learners lack prior procedural knowledge, a novel finding for example-based learning in this context. In addition to the theoretical contributions, these results support a shift in the pedagogy of advanced computer science toward worked example-based reflection, with the optimal design varying based on learners' prior knowledge.

To fulfill the promise of more effective team-programming project designs, we build on these results to evaluate two strategies sensitive to learners' prior procedural knowledge levels. The first employs reinforcement learning-based parameter optimization to determine the optimal balance of time spent on problem-solving practice and worked examples. The second employs clever instructional design to create new tasks that simultaneously promote reflection and problem-solving practice. Both designs outperform existing team programming projects and approaches that are not sensitive to learners' prior procedural knowledge levels.

Collectively, this dissertation achieves three key outcomes: evidence-backed AI-facilitated improvements to team programming project designs in advanced computer science education; theoretical advancements in example-based learning and adaptive pedagogy; and a challenge to conventional teaching practice through empirically validated alternatives. We establish a scalable technological infrastructure to enable continuous experimentation with AI-driven learning innovations, while providing actionable guidance for practitioners and researchers aiming to design adaptive, context-sensitive educational interventions. These contributions collectively redefine the pedagogical possibilities for advanced technical domains, demonstrating how AI integration and iterative experimentation can systematically bridge theory and practice in modern computing education.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Towards Measurable Learning Gains . . . . .	3
1.2	Comparing Worked Examples with Problem-Solving Practice . . . . .	6
1.3	Redesigning Programming Projects . . . . .	8
1.4	Implications . . . . .	9
1.5	Contributions . . . . .	10
<b>2</b>	<b>Theoretical Foundations</b>	<b>13</b>
2.1	Theory of Example-Based Learning . . . . .	13
2.1.1	Computer Science Education . . . . .	14
2.1.2	Ordering and Duration Effects . . . . .	15
2.2	Orchestrating Collaborative Learning . . . . .	16
2.2.1	Computer-Supported Collaborative Learning . . . . .	17
2.2.2	Online Mob Programming as Script-Based Scaffolding . . . . .	19
2.2.3	Collaborative Learning from Worked Examples . . . . .	20
2.2.4	Discouraging Unproductive Behaviors . . . . .	20
<b>3</b>	<b>Experimental Context and Project Design</b>	<b>23</b>
3.1	Experimental Context and Participants . . . . .	23
3.2	Project Design . . . . .	24
<b>4</b>	<b>Tech-Enabled Orchestration and In-Process Support</b>	<b>29</b>
4.1	Mob Programming . . . . .	29
4.1.1	Prior Work . . . . .	30
4.1.2	Roles and Role Definitions . . . . .	31
4.1.3	Mob Programming in Pedagogy . . . . .	31
4.2	Online Mob Programming . . . . .	32
4.2.1	Conversational Agent-Based Prompts . . . . .	34
<b>5</b>	<b>Towards Measurable Learning Gains</b>	<b>39</b>
5.1	Study 1 — Uptake and Size Effects . . . . .	39
5.1.1	Course Context and Design of the Scaffold . . . . .	40
5.1.2	Data Collection . . . . .	41
5.1.3	Analysis and Results . . . . .	42

5.1.4	Discussion . . . . .	46
5.2	Study 2 — Comparing against Self-Organization . . . . .	47
5.2.1	Course Context and Experimental Design . . . . .	48
5.2.2	Data Collection . . . . .	49
5.2.3	Analysis and Results . . . . .	49
5.2.4	Discussion . . . . .	52
5.3	Study 3 — Added Implicit Support and Comparison with Individual Projects . . . . .	54
5.3.1	Course Context and Experimental Design . . . . .	55
5.3.2	Data Collection . . . . .	60
5.3.3	Analysis and Results . . . . .	61
5.3.4	Discussion . . . . .	62
5.4	Study 4 — Design Overhaul and Conversational Agent-Based Prompts . . . . .	64
5.4.1	Course Context and Experimental Design . . . . .	64
5.4.2	Data Collection . . . . .	69
5.4.3	Analysis and Results . . . . .	69
5.4.4	Discussion . . . . .	71
5.5	Chapter Discussion and Lessons Learned . . . . .	72
5.5.1	Alignment with KLI Framework . . . . .	72
5.5.2	Key Design Improvements . . . . .	73
5.5.3	Final Design Alignment with KLI . . . . .	73
5.5.4	Lessons for Practitioners . . . . .	74
5.5.5	Lessons for Education Researchers Conducting Data-Driven Experiments . . . . .	75
<b>6</b>	<b>Problem-Solving versus Worked Examples</b> . . . . .	<b>77</b>
6.1	Study 1 — Problem-Solving versus Practice versus Worked Examples . . . . .	78
6.1.1	Course Context and Experimental Design . . . . .	78
6.1.2	Analysis and Results . . . . .	81
6.1.3	Discussion . . . . .	82
6.2	Study 2 — Time-Distribution between Problem-Solving and Worked Examples . . . . .	84
6.2.1	Course Context and Experimental Design . . . . .	84
6.2.2	Analysis and Results . . . . .	87
6.2.3	Discussion . . . . .	90
6.3	Studies 3 and 4 — Ordering and Time-Distribution between Problem-Solving and Worked Examples . . . . .	91
6.4	Chapter Discussion . . . . .	94
<b>7</b>	<b>Reinforcement Learning-based Formulation for Adaptive Project Design</b> . . . . .	<b>95</b>
7.1	Introduction . . . . .	95
7.2	Related Work . . . . .	96
7.3	Methodological Innovations for Educational RL . . . . .	98
7.4	Parameter Optimization Problem Formulation . . . . .	99

7.5	Experiment - Pilot . . . . .	100
7.5.1	Course Context and Experimental Design . . . . .	100
7.5.2	Results . . . . .	105
7.5.3	Discussion . . . . .	107
7.6	Deployment Experiment . . . . .	110
7.6.1	Course Context and Experimental Design . . . . .	110
7.6.2	Analysis and Results . . . . .	113
7.6.3	Discussion . . . . .	114
7.7	Chapter Discussion . . . . .	115
<b>8</b>	<b>Interleaved Worked Examples and Problem-Solving Practice</b>	<b>119</b>
8.1	Course Context and Experimental Design . . . . .	120
8.1.1	Data Collection . . . . .	121
8.1.2	Experimental Manipulation . . . . .	122
8.2	Analysis and Results . . . . .	124
8.3	Discussion . . . . .	125
8.4	Chapter Discussion . . . . .	126
<b>9</b>	<b>Discussion, Contributions, and Future Work</b>	<b>127</b>
9.1	Discussion and Implications . . . . .	127
9.2	Contributions . . . . .	129
9.3	Future Work . . . . .	131
9.4	Conclusion . . . . .	133
	<b>Bibliography</b>	<b>135</b>

x

May 3, 2025  
DRAFT

# List of Figures

1.1	Educational Experimentation At-Scale . . . . .	2
1.2	Educational Experimentation At-Scale; Step 1 — Understanding the Learner’s Context . . . . .	4
1.3	Educational Experimentation At-Scale; Step 2 — Intervention Design . . .	5
1.4	Educational Experimentation At-Scale; Steps 3 and 4 — Theory and Context-Informed Hypotheses, Experimentation and Hypothesis Refinement . . .	7
1.5	Educational Experimentation At-Scale; Step 5 — Deployment . . . . .	8
3.1	Course structure, pre-test, post-test, and delayed post-test alignment. . . .	25
3.2	Course structure, pre-test, post-test, and delayed post-test alignment . . .	26
4.1	The Cloud9 IDE Interface . . . . .	33
4.2	Architecture of the Bazaar Conversational Agent Framework . . . . .	36
4.3	Process of Output Coordination . . . . .	37
5.1	Examples of a reflection post, a prompt soliciting constructive feedback and a feedback post. Feedback post highlights instances of common ground, synthesis and encouragement. . . . .	57
5.2	Course structure, pre-test, post-test, and delayed post-test alignment . . .	64
5.3	A summary of the four roles that students are assigned to in the Online Programming Exercise (OPE) . . . . .	67
6.1	Course structure, pre-test, post-test, and delayed post-test alignment. . . .	79
6.2	Course structure, pre-test, post-test, and delayed post-test alignment. . . .	84
6.3	Example student interaction facilitated by the conversational agent during the collaborative programming phase. . . . .	87
6.4	Example interaction in the collaborative reflection phase. . . . .	89
7.1	Augmenting the Bazaar Conversational Agent Framework for Adaptive Parameter Tuning . . . . .	99
7.2	Course structure for the Cloud Computing course. The study reported in this chapter was conducted during Unit 4, which includes a collaborative project focused on constructing an inverted index in Scala. . . . .	101
7.3	Course structure for the Cloud Computing course. The deployment study was conducted in Unit 3, which includes a collaborative project focused on multi-threaded programming and consistency. . . . .	110

8.1 Course structure and positioning of experimental study — alignment of pre-test, post-test, collaborative project, and individual programming project . . . . .	122
--	-----

# List of Tables

2.1	Prior Knowledge Dimensions for Learners in Advanced Computer Science Contexts . . . . .	16
3.1	Learning objectives, examples of corresponding pre- and post test questions, information and elaboration prompts . . . . .	28
5.1	Number of Chat Contributions and File Revisions . . . . .	43
5.2	The 6 most distinguishing factors with their keywords and example turns as identified by PCA . . . . .	50
5.3	Comparison of factor loadings from PCA associated with each role, focusing on the 6 identified themes . . . . .	51
5.4	Learning objectives, examples of corresponding pre- and post test questions, information and elaboration prompts . . . . .	65
5.5	Summary of experimental studies in the design-based research iterations. . . . .	72
6.1	Learning Objectives, Corresponding Pre/Post Test Questions (Examples), Information and Transactivity Prompts . . . . .	80
6.2	Learning objectives, and examples of corresponding pre- and post-test questions and collaborative reflection prompts . . . . .	85
6.3	The boundary between collaborative programming practice and worked example discussion across tasks. Legend: a + b, where 'a' indicates time spent on collaborative programming, and 'b' represents the time spent on collaborative reflection. . . . .	86
6.4	Verbose results showing means and standard deviations of the scores at different time points. Legend: Title Line: a (b) where 'a' represents the time point and 'b' represents the total possible score that can be obtained. Content Lines: i (j) where 'i' represent the mean scores of the students in that condition for that learning objective and 'j' represents the standard deviation. . . . .	90
6.5	Problem-Solving Practice versus Worked Examples for Different Levels of Prior Procedural Knowledge . . . . .	93
7.1	Prompts presented in the discussion phase for control condition groups. The follow-up prompt is shown two minutes after the initial prompt and is the same across tasks. . . . .	104

7.2	Conflict-oriented consensus-building prompts delivered during the problem-solving phase for experimental condition groups. . . . .	105
7.3	Prompts presented in the discussion phase for experimental condition groups. The follow-up prompt is shown two minutes after the initial prompt and is identical across tasks. . . . .	106
7.4	Overview of the 2x2 Factorial Experimental Design (Deployment Study, Spring 2021 Unit 3). Note: Adaptive optimization of Project Manager prompt timing was conducted within each of the four conditions (C1-C4). Total participants: 117 students. . . . .	111
7.5	Detailed Schedule of Parameter Adaptation for Project Manager Prompt Timing (Deployment Study) . . . . .	112
8.1	Task Flow and Example Conversational Agent-Prompts Across Conditions	123
8.2	Raw test scores on conceptual knowledge for both conditions at pre-test and post-test across all four learning objectives (Scores out of 1.0; SD in parentheses) . . . . .	124

# Chapter 1

## Introduction

According to the theory of example-based learning [127], worked examples are more effective than problem-solving practice for learners with low levels of prior knowledge [78, 156]. For more knowledgeable learners, hands-on problem-solving practice becomes more effective through minimally guided application of their existing knowledge. This finding has been replicated in a variety of domains, including mathematics [170], science [126], engineering education [181], as well as introductory computer science [201]. However, it has not been explicitly studied in advanced computer science courses.

In the absence of this evidence, advanced computer science courses have come to rely heavily on problem-solving practice in the form of programming projects. These courses are aimed at preparing students for professional work. Therefore, they have perhaps been influenced by the belief that work-like practice is the best instructional strategy. Consequently, like software engineering projects in the workplace, these programming projects offer students with minimal guidance and require them to write computer programs in response to programming challenges. To wit, the following post on a popular forum for software engineers illustrates the prevailing dogma as well as the misapplication of learning science theory — “Worked examples, like other passive learning situations, can cause an ‘illusion of knowledge’ — feeling like you know and understand, but not really [9].” Although students in advanced computer science courses might possess enough prior knowledge to make minimally guided problem-solving the more effective strategy, a comprehensive investigation is necessary to make an evidence-backed recommendation.

This investigation is at the core of this dissertation to address the research question — *How do worked examples compare with problem-solving practice for conceptual learning and individual project performance in advanced computer science courses?*

To enable this comparison, we turn to artificial intelligence (AI) technologies to support the orchestration of the tasks involved in the project, as well as to provide in-process support, as needed, according to the learning intervention. Specifically, we use AI conversational agents to orchestrate in-process support for both problem-solving and worked example-based reflection in team programming projects. We show how these AI-facilitated team programming projects can be integrated into advanced com-

puter science courses to collect data and conduct experiments on the relative efficacy of different learning interventions. Finally, based on the results of the investigation, we design and deploy strategies from both learning science and AI that are most effective for learners in this context.

In effect, we undertake and demonstrate the end-to-end process of designing effective learning interventions through educational experimentation at scale, which is shown in Fig. 1.1 — We start by understanding the context of the learners. Then, based on learning science theory and similarities or differences between theoretical results and the learners' context, we design interventions that ought to be effective and derive informed hypotheses about why. We then set up experiments to test these hypotheses. Multiple rounds of experimentation may be required depending on the number of learners, the fidelity of the intervention design — adherence to protocols, the quality of delivery and consistency of execution, and its sensitivity to extraneous variables. The results of the experiments either support or help us refine the hypotheses, thus providing an improved understanding of what works for learners in this context, as well as the applicability of existing theoretical knowledge which we have thereby extended. Following an iterative process of testing refined hypotheses based on prior rounds of experimentation and running additional experiments, as needed, we arrive at a design that is most effective for learners in this context. This final design can then be deployed for large-scale impact in practice.

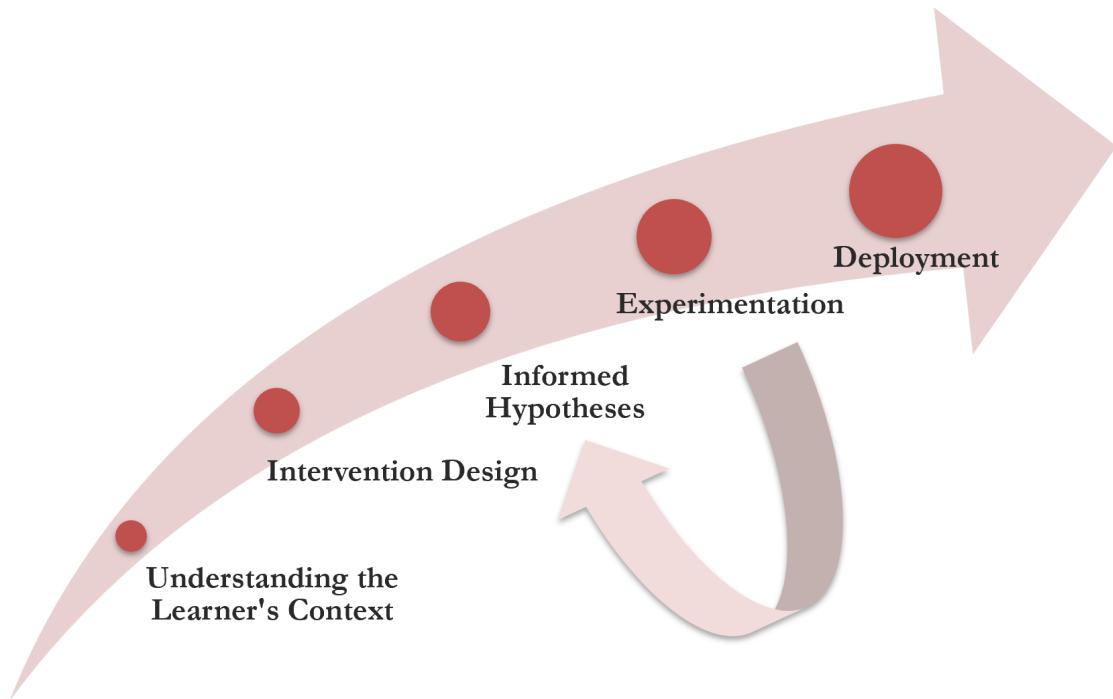


Figure 1.1: Educational Experimentation At-Scale

This approach extends established instructional design frameworks like Backwards

Design [105], ADDIE [24] (Analysis, Design, Development, Implementation, and Evaluation), and SAM [8] (Successive Approximation Model) to educational experimentation at scale.

## 1.1 Towards Measurable Learning Gains

As a precondition for conducting the proposed investigation, we need the ability to measure the learning gains from the programming project. Although this is considered a given in educational research, it is rare for projects to be individually measurable in practice. Assessments are typically performed at the end of a unit which is a sequence of lessons, or, utmost, at the end of each lesson. This is not sufficiently sensitive for experimentation. The absence of sensitive measures means that even existing project designs have not been tested for effectiveness, let alone compared to possible alternatives. Thus, by instrumenting existing projects to measure possible learning gains, we will already be adding value to advanced computer science pedagogy by collecting evidence on the effectiveness — or lack thereof — of existing project designs. To further ensure that we have a strong baseline for our investigation and that an observed effect is not due to "lousy control conditions" [156], we need a project design that produces learning gains from problem-solving practice in this context.

To produce such a project design, our first step is to refer to the details of the learners in our experimental context, as provided in Chapter 3.

In brief, we conduct our experiments in graduate-level project-based computer science courses covering advanced topics such as Cloud Computing, Data Science, and Machine Learning. These courses are composed of several units, each culminating in an individual programming project. To prepare students for this individual project, they participate in a real-time team programming project in small groups, which is the subject of our design experiments. Students are expected to learn domain-relevant concepts from the team programming project and then be evaluated on their ability to implement those concepts in practice when they work on their individual programming projects.

For the purpose of coordinating the team programming project and providing in-process support for collaborative learning, we design a new programming project interface. The interface integrates the Bazaar AI conversational agent framework [2] which was chosen for its ability to support "multi-party" conversations — conversations between multiple learners, using multiple AI support behaviors coordinated at the output to ensure consistent and non-conflicting behavior from the conversational agent. The resulting interface is integrated with the courses in our context through the Sail() platform<sup>1</sup>.

When designed to include sensitive measures of learning, this programming project interface has the potential to satisfy the precondition for experimentation that we outlined at the beginning of this section. The conversational agent additionally allows us to collect data about the learning process by capturing and responding to contributions

<sup>1</sup><https://sailplatform.org/>

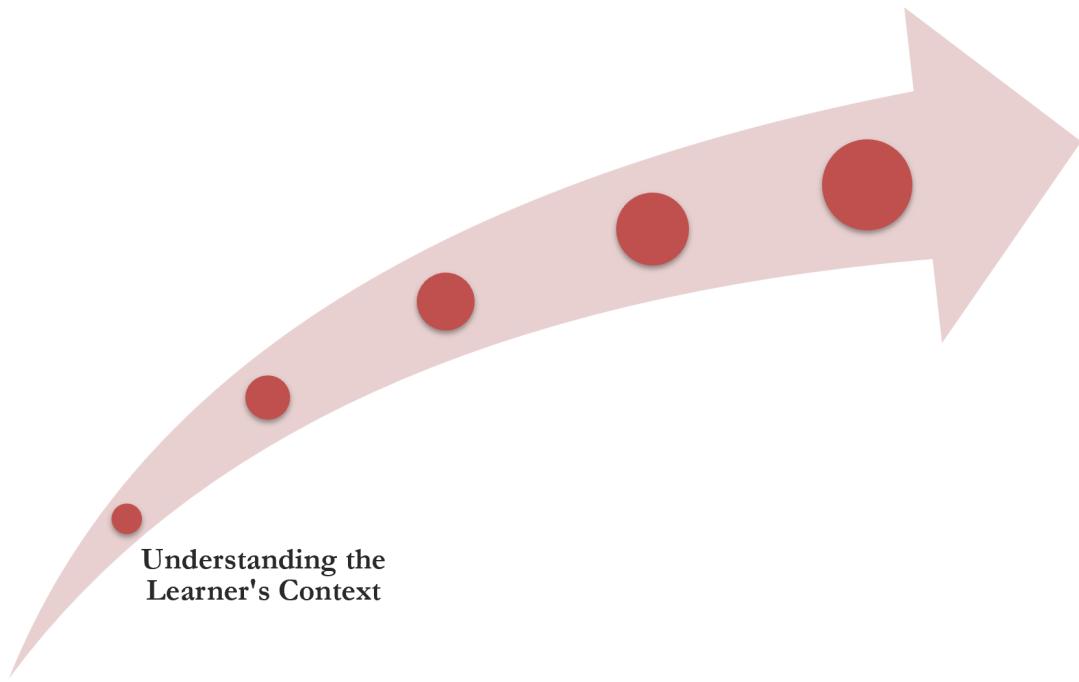


Figure 1.2: Educational Experimentation At-Scale; Step 1 — Understanding the Learner’s Context

from and interactions between learners. Integrating the programming project with the Sail() platform enables rapid design and experimental data collection iterations since updating AI behavior to support both task design and process facilitation does not require re-implementing the integration with each course. In addition, integration with the platform enables the deployment of the programming project in multiple courses offered on the platform with only topic-specific modifications. From the standpoint of education research, this is a demonstration of how learning environments can be instrumented to collect evidence about learning processes at scale and how that evidence can then be used to rapidly and iteratively improve the learning environment.

The design of AI support behaviors for the programming context, and the development of a modular team programming project to integrate with any course and remain extensible for future use are technological contributions of this dissertation. Our approach is novel in that it combines AI-supported collaborative learning with rigorous measurement of learning outcomes in advanced computer science education, an area that has seen limited research. The design of the programming project interface and the integration and extension of Bazaar to facilitate AI support for group collaboration in programming projects are described in detail in Chapter 4.

As our first finding, the existing team programming project designs do not produce measurable improvements in conceptual learning. This already demonstrates the value of instrumenting the project to measure learning gains. In addition, it makes apparent the need for a learning science-informed overhaul of the team programming project.

To redesign the team programming project not only with more sensitive learning measures, but also to have a better chance of serving as a strong baseline by producing learning gains from problem-solving practice, we build on insights from the Knowledge-Learning-Instruction (KLI) framework [88]. Over four design-based research iterations, we eventually arrive at a design that produces measurable learning gains from problem-solving practice. The details of the findings and refinements to the project design from each iteration are presented in Chapter 5. In this eventual design, we divide the project into several tasks, each focused on a specific learning objective. For each learning objective, we create corresponding questions on the pre- and post-tests that students take immediately before and after the project as a measure of domain-relevant conceptual learning (henceforth referred to as conceptual learning) from the project. The individual programming project following the team programming project can then serve as a delayed post-test measure of the ability of students to implement the concepts in computer programs. The integration of the Bazaar AI conversational agent framework to support task design and group processes, together with the learning science-informed redesign of the programming project, represents the second step in the process of educational experimentation at scale — Intervention Design.

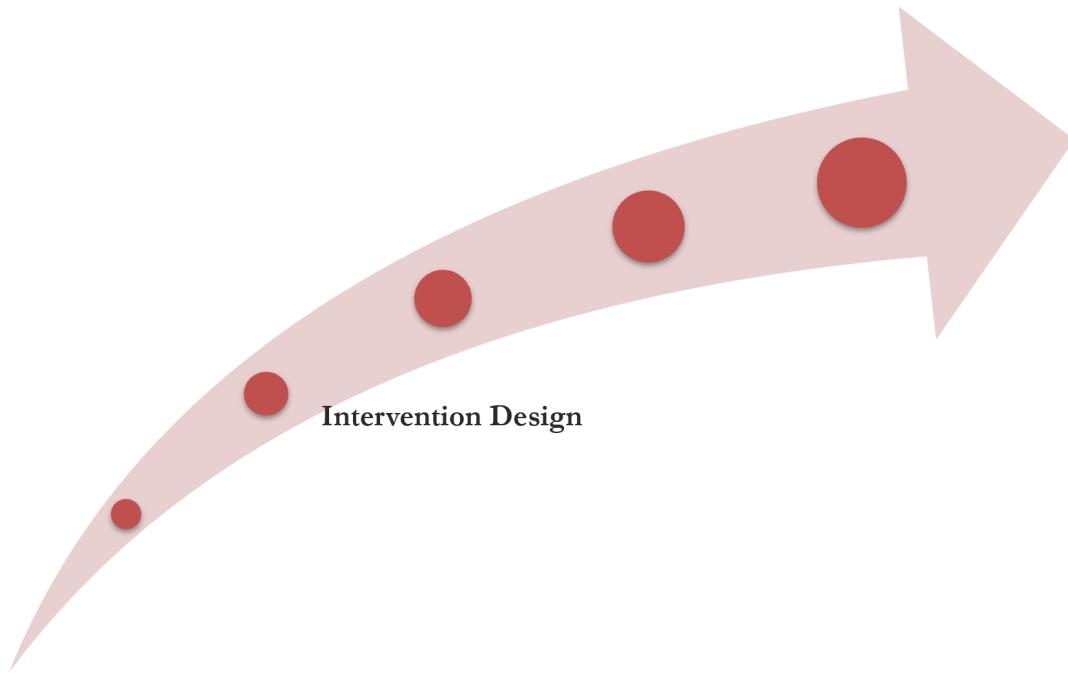


Figure 1.3: Educational Experimentation At-Scale; Step 2 — Intervention Design

The process reveals the critical distinction between learning objectives as conceived by instructors and those defined by learning scientists. Fine-grained definitions for learning objectives are required not only for measurability but also to improve the chances that targeted instruction actually produces measurable learning gains [100, 101]. Instructors often question the return on investment from what is perceived as

intensive instructional design. In this case, not only do we demonstrate the lack of efficacy of an existing project design, but we are also able to show that a learning science-informed redesign results in an effective design. Finally, instrumenting the course project in this way allows us to carry out the primary comparative investigations proposed in this dissertation on the efficacy of alternative designs using a data-driven approach.

For tasks focused on domain-relevant concepts, the refined design of the team programming project produces conceptual learning gains for learners. This constitutes the strong baseline version of the programming project design. Nevertheless, some amount of guidance, provided by the AI conversational agent in the form of reflection prompts, shows compounded benefits over and above what is achievable with problem-solving alone. Thus, testing even more guidance, such as with worked examples, would be the next logical step. However, the concern from instructors in this context is that worked examples alone, while better for conceptual learning, may leave students less able to implement those concepts in practice. Consequently, in the following chapter, Chapter 6, we compare problem-solving practice with worked examples based not only on their impact on conceptual learning, but also on individual project performance. The hypothesis from theory is that worked examples should be better for conceptual learning, since the learners in our context are novices with respect to the concepts being learned. However, the effect on hands-on task performance could be moderated by their prior proficiency in at least one programming language, which is a prerequisite for each of the advanced computer science courses in our context. This represents our hypothesis, which is informed by theory as well as factors relevant to learners in our context.

## 1.2 Comparing Worked Examples with Problem-Solving Practice

We conduct a total of four experiments. Across these, we compare three factors — the level of prior procedural knowledge, the time spent on each treatment, and the order of presentation of the treatments. We observe that the learners in this context are novices with respect to conceptual knowledge, while their level of prior procedural knowledge differs depending on the programming language they are asked to use for the project.

In the first experiment, the students have prior procedural knowledge in the programming language used. The baseline condition is the final design from Chapter 5 that maximizes the time spent on problem-solving. The experimental condition maximizes the time spent reflecting on worked examples. Both conditions first present problem-solving while keeping the total time on task constant. Consistent with the theory of example-based learning, spending more time on worked examples improves conceptual learning. More importantly in this context, this condition also results in better performance in the individual project [146].

We repeat this experiment in the subsequent semester, this time testing all combinations of time spent on each treatment and the order of treatments, for a total of 8

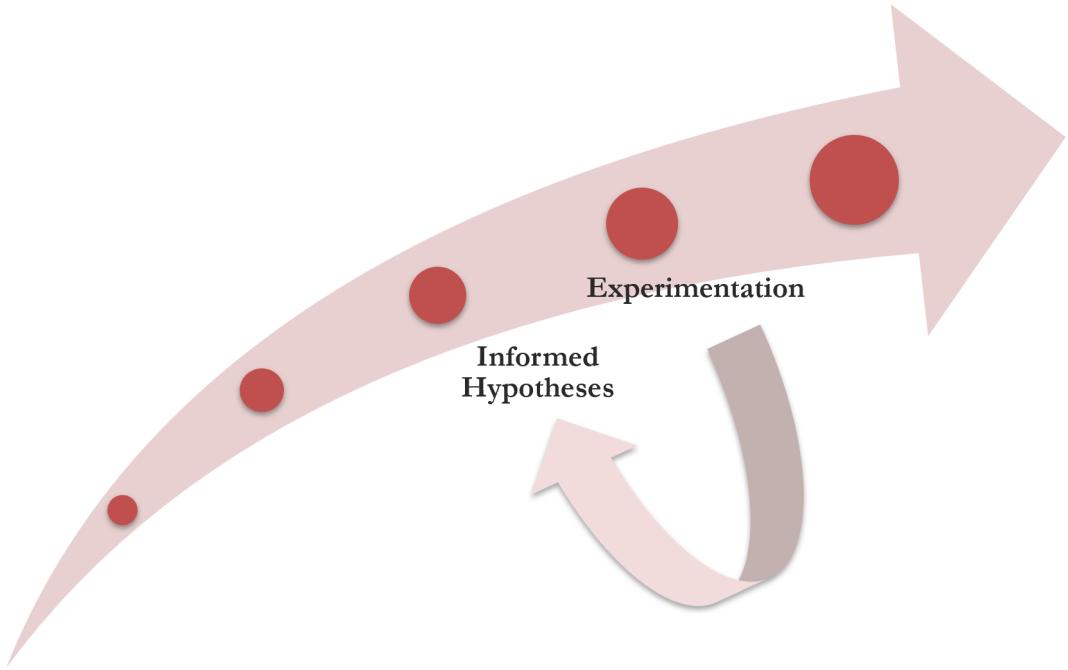


Figure 1.4: Educational Experimentation At-Scale; Steps 3 and 4 — Theory and Context-Informed Hypotheses, Experimentation and Hypothesis Refinement

conditions ( $4$  treatments  $\times$   $2$  orders). For conceptual learning, presenting worked examples first is better overall, and spending the entire time on worked examples is the best individual condition. For individual project performance, problem-solving first is generally better. Some time problem-solving and most of the time on worked examples is the best individual condition. From these two experiments, it is clear that maximizing time with worked examples is best for conceptual learning, and a little time on problem-solving is sufficient to help learners translate their conceptual understanding into computer programs.

To check if the same applies when learners do not have prior procedural knowledge in the programming language used, we compare the combinations of time spent on each treatment, for a total of  $4$  conditions [147, 149]. All four conditions first present problem-solving while keeping the total time on the task constant. Spending all the time on worked examples is best for conceptual learning from the task and incurs no performance deficit on the individual project. However, there was concern with the project design — that the time between the team programming project and the individual project might evaporate otherwise observable differences. Consequently, we repeat this study in the subsequent semester, moving the individual project to immediately after the group project. We also test the order of treatments this time for a total of  $8$  conditions ( $4$  treatments  $\times$   $2$  orders). Presenting worked examples first is better for conceptual learning, with mostly examples followed by some problem-solving being the best individual condition (though not significantly so). For individual project perfor-

mance, problem-solving first is better overall, with half the time spent on each being the best individual condition.

Therefore, even when learners do not have prior procedural knowledge, maximizing time with worked examples is best for conceptual learning. They do need some additional time for practice, but spending more than half the time on problem-solving is sub-optimal. The order of treatments has the same effect regardless of prior procedural knowledge — presenting worked examples first is better for conceptual learning, and problem-solving first is better for individual project performance. In either case, the findings support mostly worked example-based reflection, thus challenging the prevalence of problem-solving practice in today’s pedagogy of advanced computer science. The optimal design differs according to the learners’ prior procedural knowledge in the programming language used.

### 1.3 Redesigning Programming Projects

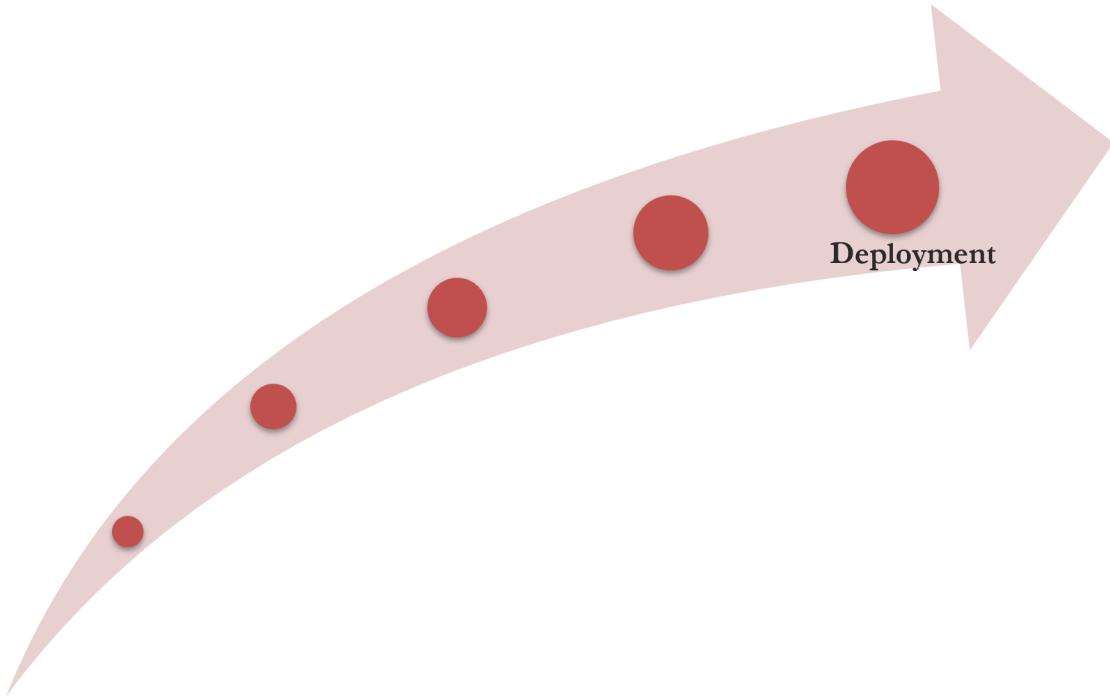


Figure 1.5: Educational Experimentation At-Scale; Step 5 — Deployment

In light of these findings, we provide two strategies for designing programming projects that take into account differences in prior procedural knowledge.

In the first strategy, detailed in Chapter 7, we use AI to customize task design to the observed behavior of the learners, specifically using a reinforcement learning (RL)-based parameter optimization to determine the optimal allocation of problem-solving practice and worked examples. This process can be re-run for learners with different

levels of prior procedural knowledge [148]. For this, we navigate concerns unique to education such as ethical implications of subjecting learners to potentially detrimental interventions during exploration as well as typical concerns about RL’s data-hungry nature. We first constrain the design of the state space based on our knowledge of the context and learning science theory. This mitigates the possibility of detrimental states in the state space. Then, we introduce a Batched Offline Update methodology, where we take a large exploration step after multiple group interactions. This derives more information from the limited data. Noisy data would render this approach ineffective. However, our new project design with sensitive measurements reduces the likelihood of this and enables the use of the Batched Offline Update method to overcome the small-data problem.

In the second strategy, presented in Chapter 8, we use clever instructional design to eliminate the need for separate problem-solving practice and worked example components. Using knowledge of the processes that problem-solving practice and worked examples stimulate, we present an “interleaved” design. Students are presented with two strategically different worked examples in pseudocode — an algorithmic description of the implementation. To choose a path to the solution, students engage in critical reflection, comparing conceptually relevant distinctions between worked examples. Then, in translating the pseudocode into computer programs, they receive problem-solving practice. We show that this strategy outperforms the presentation of treatments separately as in previous designs and can be used regardless of prior procedural knowledge levels.

## 1.4 Implications

In Chapter 9, we discuss the implications of our findings for the theory of example-based learning and the pedagogy of advanced computer science.

**Effectiveness of worked examples for conceptual learning** — The finding that spending more time on worked examples improves conceptual learning is consistent with the theory of example-based learning, as learners in this context are novices with respect to conceptual knowledge.

**Prior procedural knowledge and worked examples** — Programming knowledge can be split into syntactic (procedural) and semantic dimensions. Learners with prior procedural knowledge know the syntax well, but need the practice to determine the appropriate semantics for the new concepts they have learned. Consequently, while maximizing time with worked examples is best for conceptual learning, some problem-solving practice helps translate their conceptual understanding into computer programs.

**No prior procedural knowledge and the need for problem-solving practice** — Learners without prior procedural knowledge need time to acquire both syntax and semantics. Normally, this would not be possible within the duration of the programming project. However, as students advance in the computer science curriculum, they transfer their knowledge of fundamental programming language concepts from one programming language to another, making it possible to learn the syntax and semantics

of a new language quickly.

**Challenging prevailing pedagogy** — The findings support mostly worked example-based reflection, challenging the prevalence of problem-solving practice in advanced computer science education. They also reveal nuanced differences based on treatment order and prior procedural knowledge levels. What the results call for is a tailored approach to support learners in this context. We demonstrate that this can be achieved either with AI or with learning design.

## 1.5 Contributions

Educational data is notoriously difficult to collect. Finding learning environments that reach a large number of learners is the first challenge. Instrumenting that environment to collect the data is the second. The final and most important challenge is the quality of the data. The noisier the data, the more is needed to isolate the signal. Consequently, the first significant contribution of this dissertation is the integration of the Bazaar AI conversational agent framework into the design of a high-quality programming project, which we then integrate with multiple advanced computer science courses through the Sail() platform. This allows for continuous collection of high-quality data as well as rapid experimentation to continuously test and improve AI support for learning. The modular and extensible nature of this integration has meant that, since the conclusion of the research conducted in this dissertation, it has been extended to support multimodal interactions [191], portable interfaces [185], and the use of large language models for even more personalized AI support [111].

The first round of data collected shows instructors that existing project designs are ineffective and a learning science-informed redesign, although effort-intensive, is necessary. Extensive comparisons of problem-solving practice to worked examples then show that the latter to be more effective, challenging long-standing practice and dogma in advanced computer science pedagogy. Both of these have lasting implications for computer science educators and have already contributed positively to the experience of several cohorts of students at Carnegie Mellon University, where this research was conducted.

Finally, we propose strategies in response to these findings that produce contributions to their respective communities in AI and education. For RL, the amount of data collected in this context remains low, making it a small data problem. In addition, exploration-exploitation is uniquely problematic if it exposes learners to potentially detrimental states. We address these challenges through informed state-space design and a Batched Offline Update Model as solutions based on intimate knowledge of the context. This highlights a crucial element of education research — the details of the context and their impact on the design of learning environments.

Understanding the cognitive processes behind how problem-solving practice and worked examples contribute to learning leads to a unique instructional design solution that produces both processes through a single treatment. The continued understanding of the underlying processes that drive the observed results will produce novel interven-

tions that advance learning.

Together, data-driven AI solutions and theory-informed design represent a powerful combination demonstrated in this dissertation to produce measurable real-world impact, at scale, on hundreds of learners. More importantly, this dissertation lays the groundwork — both theoretically and technologically — for future work to expand on impact as well as understanding.



# Chapter 2

## Theoretical Foundations

### 2.1 Theory of Example-Based Learning

The primary investigation we set up in this dissertation is the comparison of problem-solving practice with worked examples in the context of advanced computer science. So long as learners are novices with respect to the concepts being learned, the literature on example-based learning [127] generally establishes that extensive problem-solving practice is inferior to the use of worked examples to positively impact their conceptual learning. Cognitive load theory [171] is typically used to explain the positive impact. Based on this theory, the identification and induction of schema — domain-specific knowledge structures [29] — from problem states is the primary function of conceptual learning from a task. Problem-solving practice may involve additional production steps that place a load on the limited cognitive resources of a learner by taking those resources away from the processes that would impact their learning the most. Consequently, the use of worked examples that eliminate those production steps and allow learners to completely focus on learning from the problem states showcased in them leads to improved learning over problem-solving practice. Worked examples have also been compared with well-supported baselines such as cognitive tutors and have been found to have a positive impact [156]. A similar mechanism explains why problem-solving practice becomes more effective as learners gain expertise. The worked examples now carry redundant information which may produce cognitive overload for these learners, thus resulting in the so-called “expertise-reversal effect” [77], that is, a reversal in the aptitude-treatment interaction as the level of the learner’s prior knowledge increases. Consequently, their limited cognitive resources are better spent on problem-solving practice because it allows them to actively apply and refine their existing knowledge, leading to deeper understanding and better schema development [78].

This finding has been replicated in a variety of domains, including mathematics [37, 120, 170], science [126], engineering education [181], as well as introductory computer science [102, 184, 201]. However, it has not been explicitly studied in advanced computer science courses. Research in computer science education has focused overwhelmingly on the introductory computer science context [40], leaving pedagogy unin-

formed when moving beyond foundational courses. In the absence of direct evidence, practice has been driven by prevailing dogma and a possible misunderstanding of general principles of learning science. For instance, advanced computer science courses are vocational in that they are meant to prepare students for work. In a misapplication of “learning by doing”, the pedagogy in this context is designed to mimic what software engineers do in the workplace. Thus, like software engineering projects in the workplace, programming projects offer minimal guidance and ask students to write computer programs in response to programming challenges. Of course, worked examples represent “doing” as well, having been designed to elicit self-explanation, reflection, and discussion among students. The difference lies in the level of guidance, with the literature showing that minimally guided instruction, such as in these programming projects, is less effective and efficient than approaches that emphasize strong guidance. The advantage of guidance begins to recede only when learners have sufficient prior knowledge to supplement with their own “internal” guidance [85]. Whether learners in this context have acquired sufficient prior knowledge to make minimally guided problem-solving the more effective strategy is an open question that requires comprehensive investigation.

### 2.1.1 Computer Science Education

From the point of view of domain-relevant conceptual knowledge in the topics being taught in these courses, such as Cloud Computing, Data Science, and Machine Learning, these students are novices. However, from the point of view of programming knowledge, the situation is more nuanced. Early research in computer science pedagogy [79, 160, 161] points to the separation of programming knowledge into its syntactic and semantic aspects. This separation is isomorphic to the separation between schema acquisition and automation referred to in the worked-example literature [183]. Syntactic knowledge combines declarative and procedural knowledge, while semantic knowledge can be further divided into conceptual and strategic knowledge [103]. Syntactic knowledge is about knowing specific facts about a programming language and rules for its use, conceptual knowledge is an understanding of computer programming constructs and principles, and strategic knowledge is the development of programming-specific versions of general problem-solving skills. Research indicates that expert programmers differ significantly from novices in the strategic aspects of programming, often viewing the process as general problem-solving, disjoint from the procedural aspects [154]. In addition, expert programmers are able to transfer the conceptual aspects of programming from one programming language to another, even when they do not know the implementation procedures of the second language. What they need the most help with are the strategic aspects of language-independent planning — transforming domain-relevant concepts they had learned into programming language concepts that they can then implement using language-dependent implementation planning. Students in the advanced computer science context fall exactly into this category. They are experts in programming, in that they have acquired declarative, procedural, and conceptual knowledge in at least one programming language. In fact, this is a prerequisite

that is enforced for all students before they can enroll in the courses in which we situate our studies (see Chapter 3. Therefore, they are novices with respect to domain-relevant conceptual knowledge, may or may not have prior procedural programming knowledge depending on the programming language of choice for the project, do have knowledge of programming language concepts which transfers across languages, and need to learn the strategic aspects of implementing domain-relevant conceptual knowledge as computer programs. These dimensions of knowledge along with the corresponding characteristics of students in the advanced computer science context are shown in 2.1. Thus, the only pertinent difference dimension is whether they have prior procedural knowledge in the programming language of choice for the project. For the sake of brevity and since conceptual programming knowledge is not a relevant difference dimension, we will refer to domain-relevant conceptual knowledge simply as conceptual knowledge going forward. Simply put, therefore, *students in this context are novices with respect to conceptual knowledge, while their level of prior procedural knowledge differs depending on the programming language they are asked to use for the project.*

Example-based learning involves less hands-on work to convert conceptual knowledge into actual programs. Thus, while helping learners with language-independent planning, it provides less practice with language-dependent implementation planning. This concern is also espoused by practitioners in this context — the concern is that while students may learn more concepts, they may be left less able to implement those concepts, once learned, in projects that follow their learning experience. Research in the introductory computer science context found that this is a possibility [201]. Therefore, in addition to comparing worked examples to problem-solving practice for conceptual learning, we also need to study their relative impact on implementation.

### 2.1.2 Ordering and Duration Effects

To round out the comparison, we also test the order of presentation of treatments for their impact on conceptual learning and implementation. The presentation of worked examples first is generally considered more effective [96, 182]. However, there is some support for problem-solving followed by worked examples when the learning elements of the task have low element interactivity — a smaller number of interconnected elements that must be processed simultaneously in working memory to achieve understanding [27], in other words, a measure of task complexity [28]. Thus, if the task is less complex or its complexity is reduced in such a way that working memory can be saved from processing too many procedures, then problem-solving first could be more effective.

For duration effects, there is plenty of evidence from studying fading — progressively transitioning from worked examples to problem-solving. Fading can be fixed or personalized based on the knowledge of when the learner hits the threshold for expertise-reversal [134]. Both of these have been shown to be more effective than problem-solving alone.

Thus, across all the experiments presented in Chapter 6, we tested the order of presentation of treatments and various options for fixed fading. In the future work chapter,

Prior Knowledge Dimension		Brief Description	Learner Expertise in Current Context
Domain-relevant	Conceptual	Conceptual knowledge in the topics taught (Cloud Computing, Data Science, ML).	Novices
Programming	Syntactic	Declarative	Knowledge of symbols and primitives (e.g., keywords, data types).
		Procedural	Knowledge of implementation procedures (e.g., syntax rules, compilation process)
	Semantic	Conceptual	Understanding of programming constructs and principles (e.g., loops, functions, object-oriented concepts).
		Strategic	Problem-solving skills specific to programming (e.g., algorithm design, code organization).

Table 2.1: Prior Knowledge Dimensions for Learners in Advanced Computer Science Contexts

Chapter 9, we talk about how recent advances in AI and large language models can enable more accurate detection of expertise-reversal and, as a result, more precise options for appropriate personalized interventions for each learner/group of learners.

## 2.2 Orchestrating Collaborative Learning

The team-based programming projects are envisioned as effective learning environments to foster collaborative learning among learners. Collaborative learning allows learners to co-construct new knowledge through productive interactions with their peers [71]. When described with the group as the unit of analysis, collaborative learning is the process of building shared understanding through interactions with others, where participants are dedicated to shared goals and problem-solving [131]. However, from

a sociocultural learning perspective, we can also look at the individual unit of analysis. Here, learners participate in meaningful peer interactions, such as explaining, questioning, justifying, and negotiating their individual conceptual knowledge while problem-solving in the collaborative learning environment [55, 69, 167]. The environment provides learners with the opportunity to externalize their mental models [81]. Then, through these interactions with peers who critically engage with the externalized mental model [23], each learner has the opportunity to update and enhance their mental model, leading to their improved conceptual understanding [58, 178, 179]. The role of technology in orchestrating and facilitating collaborative learning was recognized early, but shortcomings were already observed from simply adapting existing learning environments that were not purpose-built for collaborative learning [162]. Without support, learners faced significant hurdles in realizing the benefits of collaborative learning from high-quality knowledge exchange, negotiation, and co-creation [19]. Encouraging active participation and contributions from group members, regulating and coordinating common goals, and integrating multiple perspectives was challenging [107]. But even before that, process-related challenges such as higher levels of forgetfulness, procrastination, and social loafing than face-to-face learning, negatively impacted peer interactions [70, 92]. Learner behaviors, such as their tendency to seek quick solutions and make hasty decisions, rather than explore and critically evaluate alternatives, also undermined the theoretical benefits from being realized in practice [180]. Thus, a body of research emerged to design learning environments specifically for collaborative learning with sufficient support for learners to engage in desirable behaviors while preventing undesirable ones from manifesting.

### 2.2.1 Computer-Supported Collaborative Learning

Computer-Supported Collaborative Learning (CSCL) is based on the premise that technology can effectively facilitate collaborative knowledge construction and problem-solving. From prior research in social psychology, CSCL recognized the critical role of social interactions, including those involving conflicts [21, 45, 59], in the cognitive reorganization and development of learners [82]. Vygotsky [186, 187] highlighted that higher psychological functions first appear in social interactions, and effective scaffolding the learner within their “zone of proximal development (ZpD)” is critical for their cognitive development. Vygotsky’s experiments illustrated ways in which group cognition is a basis for individual cognition, thus advancing collaborative learning as “a basis for all human learning, not just an optional and rare mode of instruction” [163]. Collaborative learning, in general [84, 192], and the operationalization of Vygotsky’s theory with collaborative learning scaffolds that support learning in the ZpD [73], have shown many cognitive, metacognitive, and social benefits for learners. Learning with peers has shown improved levels of conceptual understanding, motivation, and engagement, producing downstream effects on course performance [38, 159] and course completion [122]. However, these positive effects require intentional pedagogy and support for peer interaction, without which peer learning has been found to fail [34].

Specific to the online context that this dissertation focuses on, early efforts to repli-

cate these results have had mixed results. The simple provision of communication technology has, predictably, proven insufficient. For instance, an early Massive Open Online Course (MOOC) that offered optional learning groups found that only 300 out of a total of 7350 participants in the course signed up for one of the 12 learning groups [91]. One explanation is that students do not fully recognize the role that social interaction is meant to play in their learning, suggesting a need for a much more intentional and tighter coupling of social interaction opportunities with course content and pedagogy [165]. Attempts to encourage unstructured discussions using real-time chat did not find improvements in student retention rate or academic achievement [33]. In contrast, chat facilitated by an intelligent agent led to an almost 50% reduction in dropout at the next time point [52, 132]. The online context also faces some unique problems that, if not addressed, are detrimental to peer learning [86]. For example, students may fail to provide feedback on the work of their peers, they may not show up for a collaboration session they signed up for, or they may quit working with their team altogether as they dropout of the course. These are some of the same reasons why students have also reported more frustration with online groups than with groups in face-to-face learning environments [159]. Therefore, designing support for collaborative learning in the online learning context is an additional challenge to tackle. Technology such as video conferencing, chat platforms, emails, and discussion boards have allowed learners to participate in asynchronous and real-time exchanges [56]. Nevertheless, the lack of structured coordination mechanisms to facilitate meaningful collaborative learning has resulted in poor learning outcomes [72].

Coordination, both social and cognitive, is crucial for effective collaborative learning. Research has shown that successful collaborative learning is based on the ability of group members to synchronize their actions, share understanding, and build on each other's ideas [44]. Stahl et al. [164] emphasized that the quality of collaborative learning is significantly enhanced when participants effectively coordinate their efforts and interactions. Supporting learners' coordination has therefore been one of the most important objectives of CSCL research [18]. In addition to coordination, "the mutuality of exchanges, the achievement of joint attentional engagement, and the alignment of group members' goals for the problem solving process" [18] are necessary to fully realize the benefits of collaborative learning. Finally, support is required to ensure that collaborative behaviors are not merely transitory either. For example, Raphael et al. [124] observed that students adopted an array of roles in their small groups that were fluid. However, when teachers returned to traditional instructional activities, students reverted to their previous interaction styles, suggesting that group collaboration norms run counter to those typically encountered in their learning [104]. Consequently, intentional pedagogical support is also required to consistently manifest behaviors associated with collaborative learning in a sustained fashion.

This pedagogical support for collaborative learning can come in various forms. The implicit variety relies on strategies such as intelligent team formation [140, 196, 197, 198] to create teams of learners more likely to engage in positive collaborative behaviors. However, implicit support may prove insufficient or impractical in many contexts, necessitating explicit scaffolding for coordination and reliable demonstration of positive

collaborative behaviors. Collaboration scripts are a form of scaffolding that help achieve this goal. Collaboration scripts guide learners through a sequence of interaction phases with designated activities and roles [118, 119] and promote conducive cognitive, social, and metacognitive processes [83]. Learners are provided with detailed and explicit guidelines in terms of what, when, and by whom certain learning activities should be performed [87]. Some representative collaboration scripts include the JIGSAW scripts [11], ConceptGrid [44], ArgueGraph [42], and WiSim [41]. The use of collaboration scripts appropriate to the task is the way in which CSCL ensures that desirable behaviors are scaffolded while preventing undesirable ones from manifesting. Similar structuring of tasks, assignment of roles, and detailing of instructional phases is required of team programming projects as well, in order to realize the benefits of collaborative learning in this context. We take inspiration from a role scaffolding paradigm (role assignment and role switching / rotation) that originated in the industry called Mob Programming [199, 204]. In addition, we use a task structuring scaffold and in-process AI support behaviors, all three of which we cast as forms of script-based scaffolding for collaboration, which we describe in the next section.

### 2.2.2 Online Mob Programming as Script-Based Scaffolding

The final design of the team programming project can be described as a task-structuring and role-scaffolding “macro”-script as defined in the Script Theory of Guidance [53]. As mentioned earlier, collaboration scripts [54, 89] specify a sequence of learning activities, learners’ roles, and other aspects of the task with the aim of promoting behaviors and triggering engagement that would otherwise occur rarely or not at all [87]. Collaboration scripts have been used to scaffold collaborative learning in several activities since they were formalized [193]. A useful separation is that of the macro- and microscript [43]. Macroscripts are external collaboration scripts that sequence the task into learning phases and/or provide high-level directives, such as role assignments, but do not provide guidance on how to act within those phases [43]. In contrast, microscripts provide explicit scaffolding for collaborative behavior within each learning phase. Macro- and microscripts have been successfully used to amplify each other’s positive effects in some prior studies [3, 109]. The task-structuring paradigm, based on best practices in instructional design of alignment between the context of the learner, the learning goal, instruction, and assessment [26], along with granular definitions of the learning objective based on the Knowledge-Learning-Instruction (KLI) framework [88] that is reached at the end of the design-based research iterations presented in Chapter 4 can be considered a macroscript. Similarly, the role assignment and rotation paradigm, based on the Mob Programming paradigm [199, 204], is a macroscript. Now, explicit in-process support in the form of conversational agent prompts intensify learning from the activity. These discourse-level prompts can be considered a microscript. Previous work used academically productive talk prompts [108] to encourage learners to elaborate on their own reasoning while challenging and extending the reasoning of their teammates [3]. They found that this form of support can significantly increase learning, but that the effect was context-specific. In particular, the effectiveness of each strategy depended

on factors such as the difficulty of the material (new versus review) and the level of knowledge of the learner (public high school versus private university).

### 2.2.3 Collaborative Learning from Worked Examples

While discourse-level microscript prompts can provide some amount of guidance, even more guidance in the form of worked examples is likely to be more effective in this context, as we have established. It is important to note that learning from worked examples hinges on being able to draw learners' attention to the relevant problem states while helping them navigate away from superfluous ones [78]. In prior work, this has been achieved by using various means such as classification of examples by common schema [176], contrasting examples, and prompt-directed self-explanation [158]. In our study, we posit that conversational agent-based prompts during a collaborative reflection can be used to direct learners' joint attention to the aspects of the problem state that are relevant to their learning. The collaborative learning context might not only allow learners to rely on the experience of all the members of the team to identify relevant problem states but, when designed appropriately, can require them to make their reasoning explicit. Then, one learner articulating their reasoning provides an opportunity for other learners in the team to challenge, extend, or integrate that reasoning with their own, resulting in transactive exchange that is associated with learning [81]. Indeed, prior work in other contexts has provided evidence for the strategic use of conversational agent-based prompts to productively impact learner behavior in a collaborative learning environment [60].

Collaboration scripts have been combined with worked examples in prior work, in an attempt to produce synergistic positive effects on learning. Although using them simultaneously did not produce any notable benefits over either scaffold separately [90], sequencing the scripts or providing them one after another, while fading the initially presented scaffold produced positive effects on dialogic mathematical argumentation skills [155]. The authors hypothesize that the positive impact of fading may be due to the fact that the two scaffolds are somewhat redundant and, therefore, the fading of one frees up the cognitive resources of the learners, resulting in additional gain from the task. It follows that in contexts where the two scaffolds can avoid redundant roles, we might observe compounded benefits from their combination.

### 2.2.4 Discouraging Unproductive Behaviors

The design of the team programming project environment is further complicated by the noted tendency of students in project-focused contexts to optimize for their project grade and adopt an efficiency orientation while sidestepping well-intentioned learning opportunities offered by instructors [68, 113]. In software engineering team projects, in particular, tasks within a larger project often require specialized expertise. Therefore, these projects can sometimes be completed more efficiently by having individuals focus on the part of each task that best aligns with their existing expertise (a divide-and-conquer strategy). However, when individuals work independently, the opportunity

for collaborative learning is lost. Consequently, these projects must be designed to explicitly discourage these tendencies in order to take advantage of collaborative learning opportunities to impact student learning. Once such an environment is successfully created, it can be instrumented with problem-solving practice or worked examples to illustrate the comparison.

To do this, we rely on a combination of task structuring and role scaffolding macroscripts, and discourse-level microscript prompts in the eventual project design. The task structuring macroscript divides the project into tasks, each of which has to be completed collaboratively prior to advancing to the next one. Within each task, students are assigned to interdependent roles with distinct responsibilities by the role scaffolding macroscript. This reduces the likelihood of free-riding since students in each role have to contribute in order to make progress on the task. The distinct responsibilities associated with the roles further reduce the likelihood of domineering and divide-and-conquer tendencies. Discourse-level microscript prompts from the conversational agent reinforce conformance to the roles in addition to providing support for reflection, elaboration, and discussion. To integrate worked examples, we modify the task-structuring macroscript to present an example for the duration of a task. The discourse-level microscript prompts now call attention to relevant aspects of the worked examples to which learners should pay attention.

Thus, positive collaborative behaviors are encouraged and unproductive ones are discouraged through the use of script-based scaffolding for collaborative learning, with appropriately designed scripts based on the context of the learners.



# Chapter 3

## Experimental Context and Project Design

### 3.1 Experimental Context and Participants

The bulk of the experiments presented in this dissertation start in the Fall 2018 semester and end in the Fall 2021 semester. This chapter elaborates on the context in which we conducted them. The very first study was in a free 6-week online Cloud Computing course offered to working professionals in the summer of 2018. This was a shortened version of a semester-long (16-week), graduate-level, online course on Cloud Computing that would subsequently be offered to the students of all of the worldwide campuses of Carnegie Mellon University (CMU), starting in the fall of that year. These professionals were all proficient programmers but were new to the conceptual aspects of Cloud Computing. The main premise behind using this course was to begin experimenting with team programming project designs that would produce measurable learning gains. It was also chosen because we were adapting the Mob Programming paradigm from industry to scaffold the team programming project. The industry audience, likely more familiar with the paradigm, may not have as much overhead in learning the scaffold, allowing us to focus on refining the instructional design. This experiment was the first in the series of design-based research iterations described in Chapter 4.

The subsequent experiments in that series were conducted in the semester-long version of the Cloud Computing course. The course offers hands-on experience with the applications of Cloud Computing concepts using three leading cloud computing platform providers – Amazon Web Services, Microsoft Azure, and Google Cloud Platform. It sees participation from between 100-200 students depending on the semester, distributed across at least four of Carnegie Mellon University’s campuses in Pittsburgh, Silicon Valley, Rwanda, and Adelaide. The majority of the participation comes from the Pittsburgh campus. Depending on the semester, there may be occasional participation from partner universities, such as Sun Yat-sen University through CMU’s Open Learning Initiative (OLI)<sup>1</sup>. Although the course itself is fully online, the student sub-

<sup>1</sup><https://oli.cmu.edu/>

populations are co-located on each campus and can meet outside of the course. This has some design implications for the design of the project, especially for the process of forming teams and administering the project. For instance, internationally distributed teams will have to find a time that works for all participants in order to do these programming projects together, in real-time. This creates overhead for the course instructors as well as a sub-par collaborative learning experience for the students. These were findings from our previous research in this context [138, 139, 175]. Consequently, we enforce co-location for all team members of a project group. The design implications from the specifics of the context are highlighted alongside descriptions of the experimental designs, where relevant. Almost all of the students enrolled in these courses are master's students. The prerequisite is that they are proficient in at least one programming language. This allows us to make certain inferences about the prior levels of programming knowledge that these students possess. Specifically, they are experts in the declarative, procedural, and conceptual aspects of at least one programming language.

At the end of the design-based research iterations in Chapter 4, we successfully produce a programming project design that produces meaningful and measurable learning gains in this context. That design is integrated with the Social and Interactive Learning Platform, abbreviated as Sail() <sup>2</sup>. The platform enables the projects to be offered in more than just the Cloud Computing course. Importantly, it enables rapid iteration on the design and AI facilitation without requiring reintegration with each semester of the course or additional courses onboarded to the platform. The first course added to the platform after the Cloud Computing course was a foundational course on Computational Data Science. This was the flagship course of the Master's program in Computational Data Science offered at Carnegie Mellon University's Language Technologies Institute. As such, the courses are structured very similarly and target a similar audience of learners. The Data Science course is an additional subject of the remaining experiments presented in Chapter 6 – Chapter 8. Since the completion of these experiments, the Sail() platform has been integrated with community college and vocational education courses, proving the value of the integration not only for experimentation, but for the deployment of successful designs to several contexts at scale. We will discuss this in more detail under Future Work in Chapter 9.

## 3.2 Project Design

Both courses are structured around several project-based units. Each unit has several sub-units and culminates in a large individual project that has assessment components to evaluate achievement in that sub-unit. In the final design, students, in teams of 4, work on their team programming project in real-time, to prepare for this individual project.

The Cloud Computing course has five such units, with the experiments conducted in either the third or the fourth unit. The third unit uses Java, which all the students

<sup>2</sup><https://sailplatform.org/>

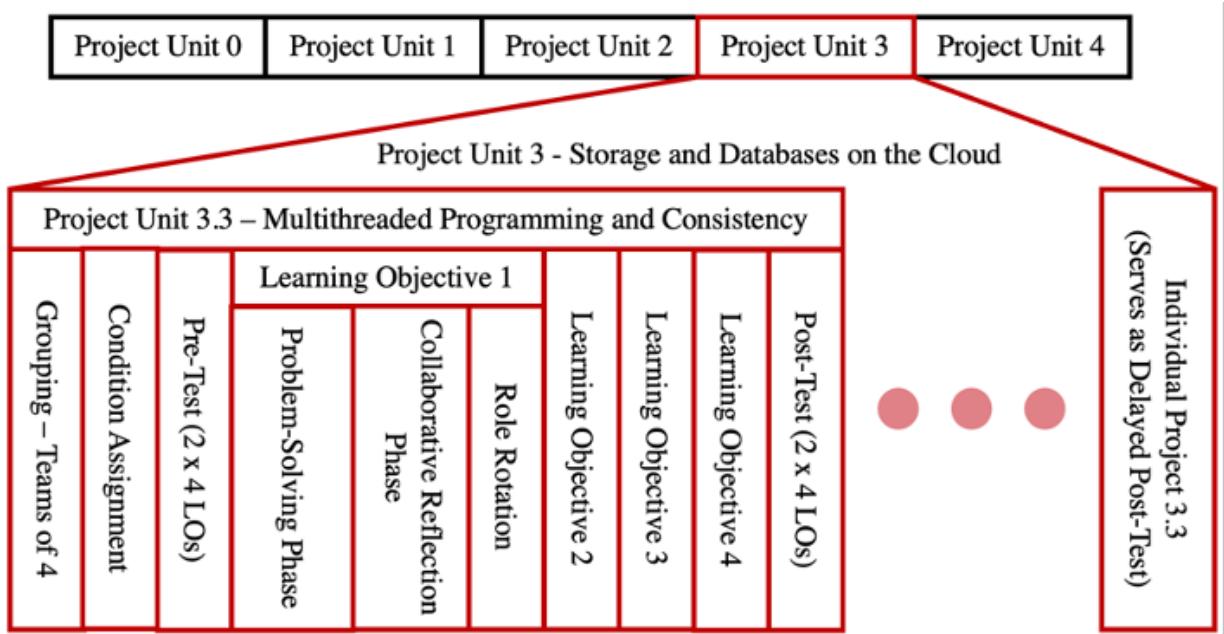


Figure 3.1: Course structure, pre-test, post-test, and delayed post-test alignment.

were familiar with. Hence, they have prior knowledge in this language. The fourth unit uses Scala for which students do not have prior procedural knowledge. Declarative knowledge of Scala is obtained from interactive primers provided prior to the team programming project. From the theory referenced in Chapter 2, we know that conceptual knowledge transfers from one programming language to another for these “expert” programmers. Thus, prior procedural knowledge remains the key difference dimension in our experiments. The difference in the requirements for the third and fourth units allows us to test the aptitude-treatment interaction between prior procedural knowledge and the use of worked examples in a between-subjects manner. This comparison between problem-solving practice and worked examples, including the order of presentation of the treatments, for students with and without prior procedural knowledge, is presented in Chapter 6.

A summary of the structure of the course and the location of the experiment within the third unit of the course is shown in Figure 3.1 and a summary of the experiment in the fourth unit is shown in Figure 3.2. For example, within the first sub-unit of the fourth project unit of the course, students work in our collaborative learning environment on a programming project to build an inverted index using the Scala programming language.

We realized very early in the design process that the lack of learning objectives at the right level of granularity and the learning measures corresponding to those granular learning objectives significantly affected our ability to measure learning gains from the project. More importantly, this lack of specificity also reduced the efficacy of the project. Given the intensive investment that would be involved in a redesign, we had to first demonstrate to course instructors that the existing project designs were ineffective and only then engage in the redesign. For the final design, we turn to best practices in

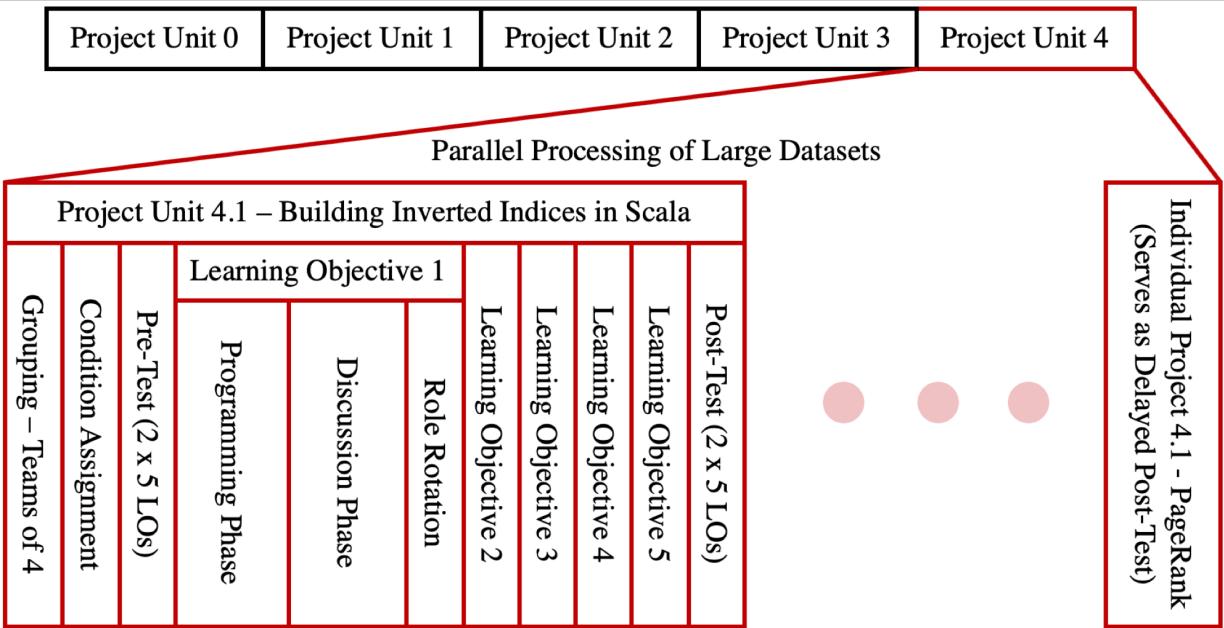


Figure 3.2: Course structure, pre-test, post-test, and delayed post-test alignment

instructional design that emphasize the need for well-defined learning goals. Then, for a given learning goal, the instruction must be positioned to explicitly target that learning goal, and the assessment should measure what was targeted [26]. This process is a distillation of several instructional design frameworks, such as Backwards Design [105], ADDIE [24] (Analysis, Design, Development, Implementation, and Evaluation), and SAM [8] (Successive Approximation Model).

To define measurable learning goals at the right level of granularity, we turn to the Knowledge-Learning-Instruction (KLI) [88] framework. Accordingly, for our example of building an inverted index using the Scala programming language, we might have the following knowledge components –

- Conceptual knowledge:
  - Understanding of inverted index structure and purpose
  - Principles of functional programming in Scala
- Procedural knowledge:
  - Scala syntax and data structures
  - MapReduce paradigm for distributed computing
  - Algorithms for tokenization and index construction
- Declarative knowledge:
  - Scala language features (e.g., case classes, pattern matching)
  - Spark API for distributed computing

These imply the following specific and measurable learning objectives (LOs) using the

KLI framework.

- Remember and Understand
  - Define an inverted index and explain its purpose in information retrieval systems.
  - Describe the basic structure of a Scala program and its functional programming features.
- Apply
  - Implement a function to tokenize input text into individual terms using Scala's string manipulation methods.
  - Create a case class to represent document-term pairs with appropriate fields.
- Analyze
  - Compare the efficiency of different data structures (e.g. Map, Vector) for storing the inverted index in Scala.
  - Analyze the time and space complexity of the inverted index construction algorithm.
- Evaluate
  - Assess the impact of removing common "noisy" words on the quality and size of the inverted index.
  - Evaluate the scalability of the implemented inverted index solution using various input sizes.
- Create
  - Develop a Spark application that builds an inverted index from a large corpus of text documents using Scala.
  - Design and implement a query function that efficiently retrieves relevant documents based on input terms using the constructed inverted index.

Now each of these imply a different instructional approach. For the "Remember and Understand" LOs, we use interactive primers with self-assess questions prior to the team programming project. The "Apply", "Analyze", and "Evaluate" LOs can be addressed either through worked examples, guided practice, or minimally guided problem-solving in the team programming project context, depending on the outcomes of our experiment. The "Create" LOs are best addressed in the individual programming project at the end of the sub-unit.

Following this recommendation, we divide the team programming project into five different tasks that target five LOs. Each LO is assigned two multiple-choice questions each on a pre-test that students take immediately before and a post-test they take immediately after. This provides a proximal measure of student learning from the project. The tasks of the team programming project are then aligned with those of the individual project which then serves as a delayed post-test. Table 3.1 shows the LOs and examples of pre- and post-test questions corresponding to each task, and Figure 3.2 shows the

position of pre-, post- and delayed post-tests within the course.

#	Learning Objective	Example Pre/Post Test Question	Information Prompts	Elaboration Prompts (Example)
1	Writing OS-aware code.	What needs to be kept in mind when writing OS-aware code?	@Researcher, what is the advantage of writing OS-aware code like you did here?	@Driver, How would you improve the implemented approach? The Researcher can chime in too.
2	Need for pre-processing	Whitespace characters are removed while building the index. Why is this necessary?	Ok, @Navigator, can you say why these preprocessing steps are necessary before building the inverted index?	@Researcher, do you agree with this approach? @Project Manager, what do you think?
3	“Map” for word level count.	Which of these is true about combining word counts from different parts of a document?	@Driver, can you tell us how the word level counting is going to help us parallelize?	@Project Manager, do you think there is better implementation of this function? @Driver can chime in too about why.
4	“Reduce” for collating counts within document.	Which Spark code snippet produces an RDD of tuple (word, word frequency)?	@Researcher, can you think of another way to do the reduce?	@Navigator, what trade-offs do you think you made in this function implementation? @Driver, can you provide some insights too?
5	Collating across documents for inverted index.	What needs to be done to build a case-sensitive index?	@Driver, what did the groupBy operation achieve here?	Ok @Researcher, is there a better way to implement this? @Navigator, can you provide insight too?

Table 3.1: Learning objectives, examples of corresponding pre- and post test questions, information and elaboration prompts

At the end of the design-based research iterations in Chapter 4, we successfully produce a programming project design that produces meaningful and measurable learning gains in this context.

This project design produces conceptual learning gains from problem-solving and, therefore, serves as a strong baseline for subsequent comparison experiments with interventions that provide additional guidance. Before detailing the design-based research iterations that helped us arrived at this project design, in Chapter 5, we will look at the technology that enables orchestration of the team programming project and in-process support using AI conversational agents in the next chapter.

# Chapter 4

## Tech-Enabled Orchestration and In-Process Support

We established in Chapter 2 that intentional scaffolding is required to help learners consistently and sustainably manifest behaviors associated with collaborative learning while avoiding pitfalls. To that end, we design a scaffold for team programming projects that we will first describe. Then, we will see how AI technology can be used to orchestrate the team programming project and deliver these scaffolds. In doing so, we will also instrument the project to capture process data.

Prior work echoes this need for the process to be instrumented and captured as well. Ju and Fox [76], for example, highlight the need to instrument learning environments to capture learner behaviors during collaboration rather than just the end product. Instructors are known to be largely unable to make these assessments since they are not present during most of the group work [61], and are also known to fall prey to specific cognitive biases when they form impressions from what they do observe [63]. Process data, in addition to technology to orchestrate the project and deliver the scaffold, unlocks a virtuous cycle for explicit in-process AI support to intensify the learning gains from the project.

### 4.1 Mob Programming

We draw inspiration and conceptual foundations from industry to motivate a potential solution to the obstacles faced in realizing the benefits of collaborative learning in practice. To do this, we build on the industry practice of Mob programming [199, 204]. In an instructional context, we cast the Mob Programming paradigm as a form of script-based scaffolding for collaborative learning. We call the resulting paradigm for collaborative project-based learning in computer science — On-line Mob Programming (OMP).

Mob Programming grows out of Pair Programming and is a group activity in which 3-6 participants assume different roles to collectively contribute a solution to a programming challenge. In this way, cognition is distributed and group members with different abilities are able to contribute in different roles while benefiting from the support of the

group. OMP first ensures that group work is conducted online, where it can be instrumented to collect data and provide data-driven support. Specifically, group processes and progress are scaffolded by AI conversational agents. AI conversational agents have been used in prior work to offer dynamic, context-sensitive support for collaboration in the middle of text chat-based interactions, thus allowing for in-process feedback and scaffolding [93, 94, 172, 173]. These forms of support have been shown to improve learning, knowledge integration, and team performance in facilitated collaborative activities [3, 48, 190]. OMP additionally provides the benefit of a structured collaboration that manages group processes up to relatively large group sizes, anywhere from 3-6 learners. The defined roles and role-switching allow learners to contribute to the task from different perspectives while being supported by the conversational agent. Importantly, role assignment discourages divide-and-conquer tendencies that learners in this context fall for due to an efficiency mindset.

#### 4.1.1 Prior Work

The Mob Programming construct is poorly studied in academic research, especially for pedagogical uses. Buchan and Pearl [25] report on a software company team’s experiences over 18 months of in-person Mob Programming. They report benefits particularly with respect to the broad knowledge and understanding that team members gained about the project as a whole. Kattan et al. [80] present three case studies on Mob Programming. Each study consists of a team in an academic setting working on open-source code. In general, they report that there was unanimous approval for Mob programming during team retrospectives. Malmgren [99] presents an experience report describing how she participates in “Remote Mob Programming”. Barring these experience reports, references to Mob Programming for pedagogy are scant.

In contrast to Mob Programming, there has been extensive research into Pair Programming. Some Pair Programming work relevant to the OMP context is referenced below. Harsley et al. [65] use an intelligent tutor to assist with Pair Programming. Rodríguez et al. [130] found that more active participation from the “Driver” — the participant writing the code and in the metaphorical driver’s seat. Encouraging active participation is an important requirement for collaborative learning and may therefore also have implications for the design of the OMP scaffold. Other researchers have studied distributed Pair Programming [15, 39, 169, 177] where pairs are not physically co-located. In addition, researchers have investigated ideas adjacent to Pair Programming with more than two participants.

Saros [135] is a platform that supports up to five participants in what they call distributed party programming. However, Saros does not provide guidance on how participants should interact during a session. Nguyen et al. [114] present EduCo, a web-based collaborative learning environment. They report supporting 40 students who participated in a single session.

### 4.1.2 Roles and Role Definitions

In Mob Programming, participants typically rotate through the following roles —

- Driver — A single participant who converts high-level instructions from the Navigator into code.
- Navigator — A single participant who makes decisions based on discussions with the mob and communicates that to the driver to be implemented into code.
- Mob — A group of participants who consider and deliberate between multiple alternative implementations, ultimately informing the decision of the Navigator.
- Facilitator — A single participant (optional) who observes and intervenes when necessary, such as to indicate when roles are switching and to keep the activity progressing.

The rotation of roles gives participants the opportunity to experience how group processes change when leadership changes within a group. Each participant will experience all the roles over the course of a single Mob Programming session, getting an opportunity to contribute, as well as observe different perspectives and approaches to solve problems.

In the Navigator role, a participant is responsible for making decisions about the next step in the implementation. They will ask for input from the mob, decide on the next action, and provide directions to the driver.

In the Driver role, responsibilities include taking directions from the navigator and translating those into code. The Driver role allows a participant to focus on writing syntactically correct code, while the navigator worries about the overall direction.

In the Mob role, a participant is responsible for considering alternative directions, understanding the current state of the code, and providing input to the navigator on how the code can be improved.

Along the programming knowledge dimensions, the Navigator and Mob members are engaging in strategic language-independent planning. Whereas, the Driver and Navigator engage in language-dependent implementation planning which the Driver ultimately implements.

### 4.1.3 Mob Programming in Pedagogy

When applied in a pedagogical context, this synchronous learning supports learners at different skill levels or those with complementary skills to scaffold each other. The group context creates pressure to perform and yet balances that pressure with support from the team. The learners are exposed to different perspectives for solving problems, building solutions, experimenting, debugging, and writing readable code. They are forced to externalize their thinking, which provides the opportunity for knowledge gaps to be revealed and addressed. They also have the opportunity to observe knowledge and expertise in action as they learn.

The concern with collaborative group activities is always that group processes come in the way of learning and productivity, and without structure, they can descend into

chaos. In addition, increasing group size can exacerbate instances of social loafing [121] (free-riding) and losses due to increased difficulty in group coordination. Without guidance, learners can also end up choosing suboptimal strategies, such as working on tasks or roles that they are already good at, instead of taking the opportunity to learn. Finally, group work can be dominated by the most outspoken members [74].

Mob Programming is pedagogically positioned to solve these issues. First, providing a structure around the collaboration and helping learners understand how this structure contributes to their learning and success in the problem can stem group coordination difficulties at the outset. Second, assigning learners to roles and periodic rotation of roles can prevent the adoption of suboptimal strategies. Moreover, role assignments can solve the problem of freeloading by introducing an element of social pressure but at the same time balancing that pressure with support from team members in other roles.

## 4.2 Online Mob Programming

The primary technological infrastructure needed to bring the offline version of Mob programming online is a terminal that can run collaboratively composed code and a means of communicating around the code. However, as mentioned previously, simply providing the technology without intentional design that can support collaboration leads to failure [86]. We therefore adopted a smaller iterative design process over a series of pilot studies, pausing after each to reflect on what worked and what did not work. We then use the insights from the investigation of this space of possibilities to design the final OMP setup for our study of measurable learning gains from the programming project. These studies are described in Chapter 5.

One locus of investigation was the communication channel. Although audio can lead to a much smoother interaction and coordination experience, it comes at two important costs. First, the audio channel is harder and more resource-intensive to capture and instrument for the purpose of analyzing the interaction between learners. The text channel is readily instrumented because it completely captures the communication among learners around the code. Second, the audio channel could disadvantage certain kinds of learners.

Prior work has found that audio channels place a higher cognitive load on learners compared to text channels, therefore disadvantaging learners with low or intermediate spoken proficiency [133, 168]. Furthermore, audio channels have been associated with higher anxiety levels among learners than text channels [64, 153]. It has also been observed that while audio channels provide a heightened sense of social presence to online learners and minimize the chances of misinterpretations because of the use of voice and tonal cues, text channels may be entirely more appropriate for tasks that require learners to explore dissonances among ideas and negotiate opinions, which is the case in Mob Programming. These tasks typically require that learners challenge each other's opinions and ideas [98] which is better supported by the text channel as it provides more time to structure responses [66].

The second locus of investigation was the collaborative coding environment. Al-

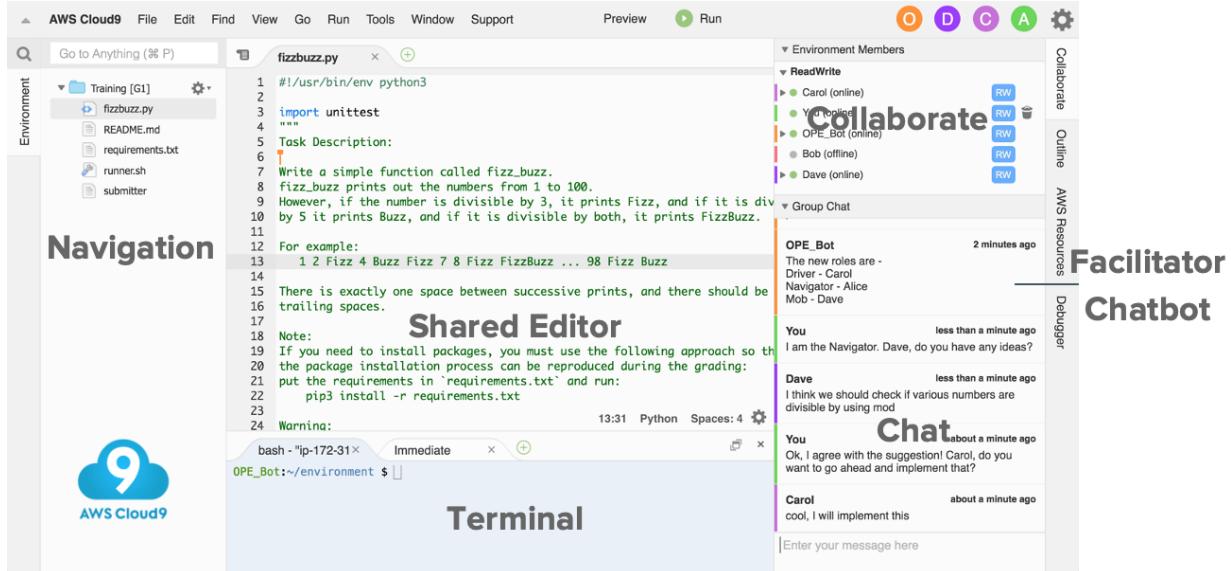


Figure 4.1: The Cloud9 IDE Interface

though tools like Coderpad<sup>1</sup> provide support for collaborative coding and log edits made to the code for later retrieval and analysis, they do not have a built-in communication channel. This necessitates the use of another application for communicating around the code in real-time. Therefore, learners would have to switch between the two applications, leading to coordination issues, switching costs, and an overall suboptimal interaction experience. This further had the unintended consequence of slowing down the entire group, resulting in the learner in the driver role having to wait for the group to come to a consensus before translating that into code. Coupled with the relatively frequent switching of roles, this meant that not much code was written in the amount of time allotted for the exercise. The learning goals were not met due to the suboptimal interaction experience and the productivity goals were not met due to coordination issues. This meant that the ideal collaborative environment would provide for both collaborative coding and support for communicating around the code in the same environment without the need for context switching.

We choose the Cloud9 IDE<sup>2</sup>, which includes editor, terminal, text chat, and file navigation, all on one screen. The collaboration pane above the chat includes some social presence tools allowing users, at any given time, to determine which files and locations within files other users are looking at. Cloud9 being a part of the Amazon Web Services (AWS) infrastructure allows the setup to scale to a large class and to service many simultaneous mobs. In addition, contributions to the code and chat could be logged along with user timestamps, which is conducive to analysis. A screenshot of the Cloud9 IDE is shown in Figure 4.1.

The final locus of investigation was the use of a conversational agent to support

<sup>1</sup><https://coderpad.io/>

<sup>2</sup><https://aws.amazon.com/cloud9/>

collaboration between learners who participated in the mob. Specifically, the role of the facilitator, which if assigned to a learner would mean either that the learner cannot fully concentrate on the task at hand or would be idly occupied by time-keeping and other coordination tasks providing no pedagogical benefit. Thus, a conversational agent facilitator, based on the open source Bazaar framework<sup>3</sup> [3] was integrated into Cloud9 to handle the timing and sequencing of the activity by informing the learners when they would need to switch and what their new roles would be. The introduction of the agent also allowed for instructions about the use of the Cloud9 IDE and the activity to be introduced more naturally as a part of the conversation rather than as a part of an instruction file, and opened up the possibility of more dynamic context-sensitive conversational support for learners and their roles in subsequent experiments.

#### 4.2.1 Conversational Agent-Based Prompts

Throughout the design-based research iterations, various types of conversational agent-based prompts were implemented to support and enhance the collaborative programming experience. These prompts played a crucial role in structuring the activity, facilitating reflection, and promoting learning.

The most important function of the Bazaar conversational agent framework is to deliver these prompts intelligently without the behaviors clashing with each other. For milestone-based advancement of the task, for example, the role-switching function listens for an event that is broadcast at the end of a task — say — from a test case passing. In response, it triggers the role-switching prompt which is sent to the output coordinator. Provided other actions proposed at the same time are lower priority, or there are no other proposed actions or blockers in the output coordinator queue at the same time, the role-switching prompt is passed and sent to the learners. Bazaar accomplishes this with the use of events that are triggered for each relevant action made in the programming project interface, and event listeners that are “listening” or subscribed to these events in order to respond to them when they are published to the event queue. The output actions are then sent by each event handler to the output queue from where they are processed in time order as well as priority order. For example, In case time in the programming portion of the activity runs out, the phase-switching prompt is pushed to the front of the queue, and the rest of the input and output queues are cleared. No subsequent actions from irrelevant event listeners enter the queue because those events are no longer triggered. Consequently, from the point of view of the user, only a sequence of relevant prompts is sent at the appropriate times without clashes between them [2, 3]. Without centralized coordination, therefore, Bazaar manages multi-party, multi-agent coordination. Attributes such as priority, timeout, and lingering advisors are attached to events that are independently proposed by actors. Events are able to be proposed independently by actors who rely on the output coordinator to ensure behaviors do not clash and learners receive a coordinated experience at the output. The architecture of the Bazaar conversational agent framework is shown in Figure 4.2. The

<sup>3</sup><https://github.com/DANCEcollaborative/bazaar>

process of output coordination is demonstrated with an illustrative example in Figure 4.3

The following types of prompts were utilized —

### **Role Scaffolding Macroscript Prompts**

The conversational agent provided prompts to support the Mob Programming structure by —

- Assigning roles to team members at the beginning of each task
- Providing instructions on role responsibilities
- Prompting role switches at appropriate intervals
- Initially, role switches were time-based, occurring every 7-8 minutes. In the final iteration, the strategy evolved to milestone-based switching, with roles changing after the completion of each task and its associated collaborative reflection.

### **Task Structuring Prompts**

To ensure that teams progressed through tasks efficiently, the agent provides —

- Bottom-out hints to hasten task completion if a team exceeds the allotted time (e.g., 14 minutes per task).
- Warnings before and notifications when it was time to move on to the next phase or task.

### **Discourse-level Microscript Prompts**

- Information prompts — Designed to focus discussion on specific knowledge related to the learning objectives
- Elaboration prompts — Aimed at encouraging deeper reflection and exploration of alternative perspectives
- Collaborative Reflection prompts — Findings from investigating the above two designs led to the creation of the collaborative reflection prompts which encouraged learners to construct arguments, supported them in warranting their claims, and explicitly prompted members to build on each other's arguments.

These prompts were presented during the collaborative reflection phase after the completion of the task. The agent demonstrated some intelligent detection by waiting for the ongoing discussions to conclude before presenting the prompts. The more interesting forms of support involved detecting arguments expressed in the conversation and triggering support behaviors such as elaboration or transactive exchange triggers, as appropriate, in response to those arguments.

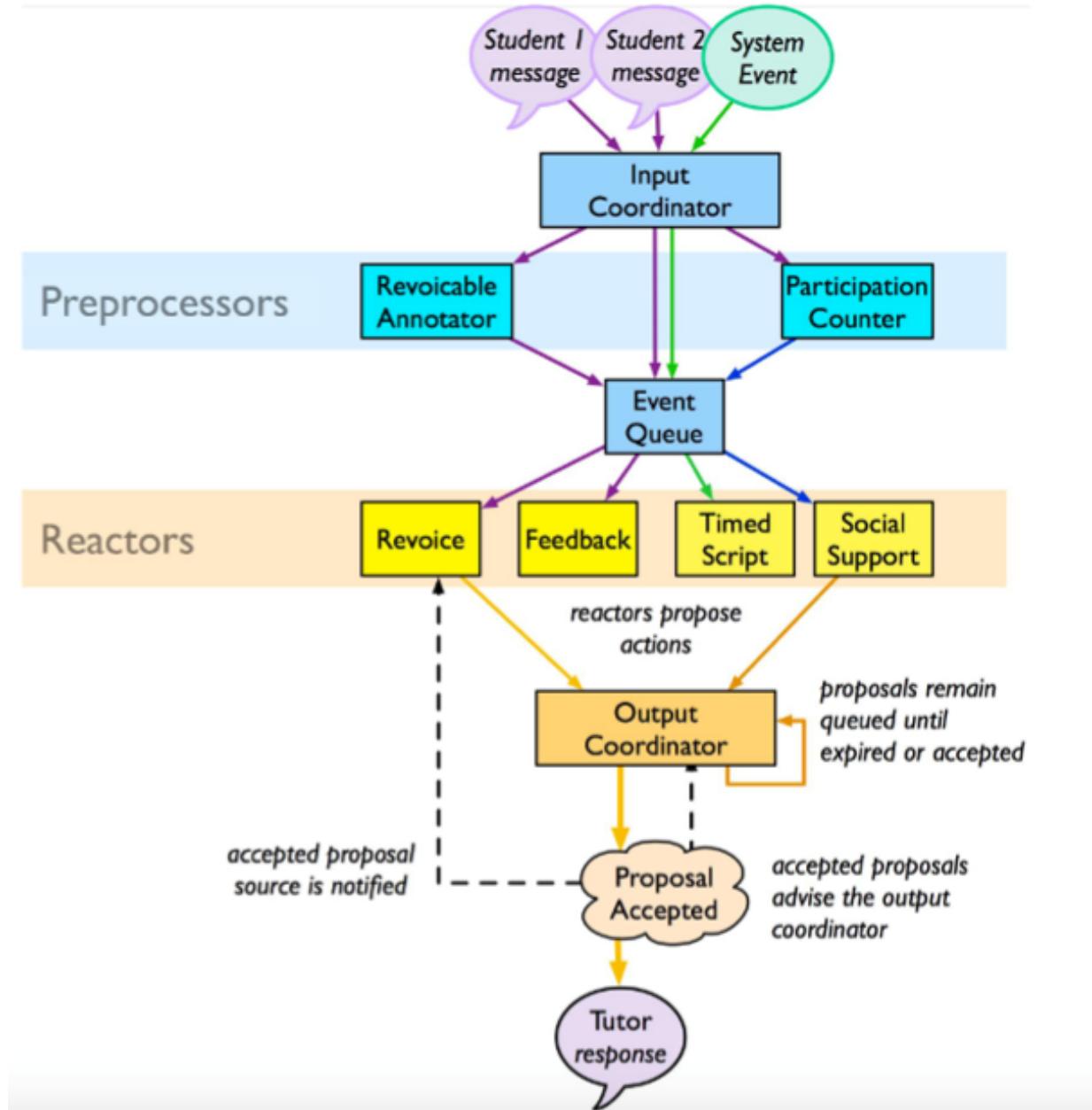


Figure 4.2: Architecture of the Bazaar Conversational Agent Framework

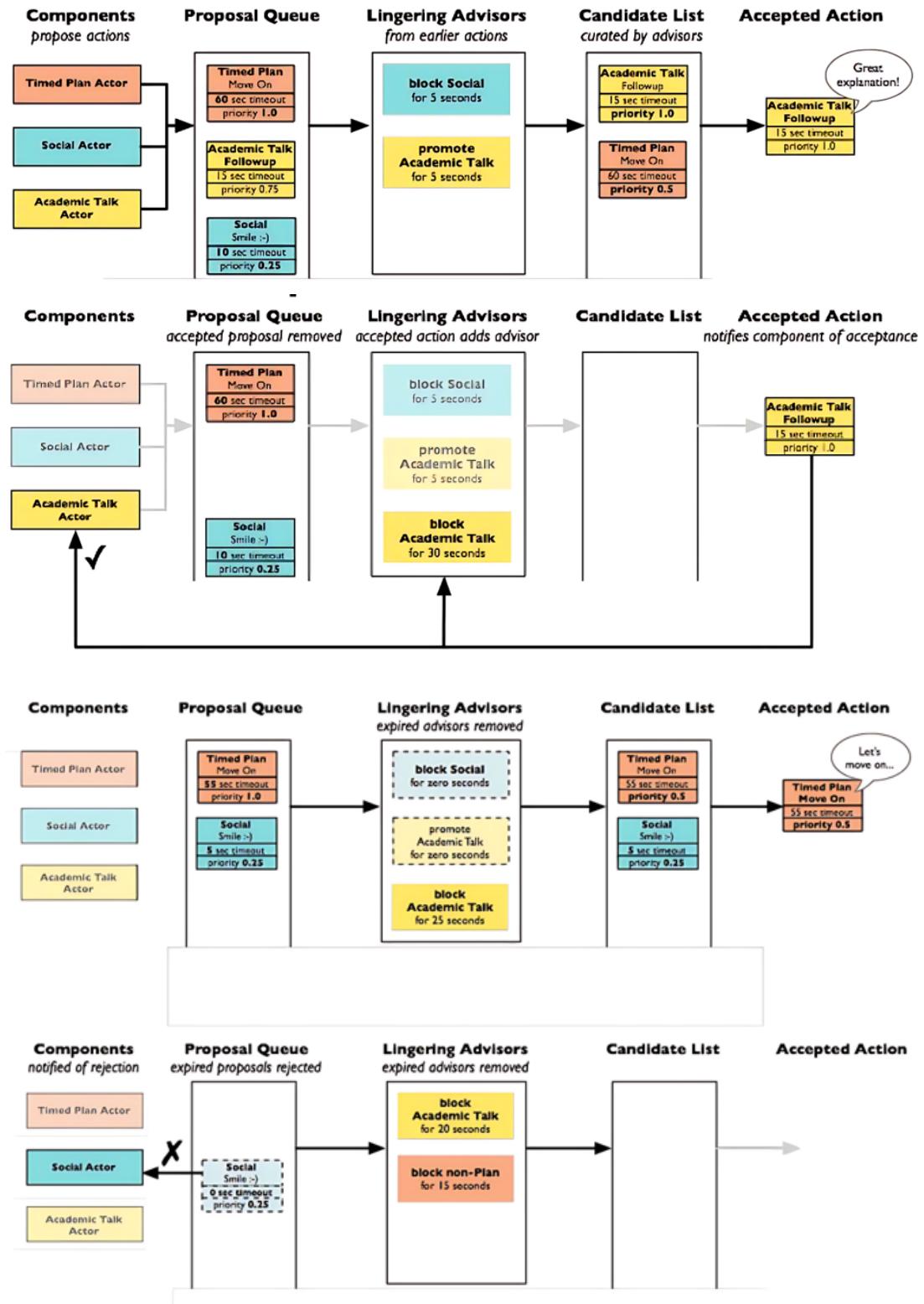


Figure 4.3: Process of Output Coordination

## RL-based Parameter Optimization for Adaptive Prompt-based Scaffolding

- Adaptive Prompt Selection — RL algorithms can be used to dynamically select the most appropriate prompts based on the current state of the collaborative programming session. The agent can learn which types of prompts (information or elaboration) are most effective for different learning objectives, team dynamics, or stages of the task.
- Timing Optimization — An RL model can be trained to identify optimal moments to introduce prompts, maximizing their impact on learning without disrupting productive collaboration. This includes macro decisions such as the duration split between reflection and problem-solving.

Both of these premises are born out in Chapter 7 which describes the use of a Reinforcement Learning (RL)-based parameter optimization to adaptively change conversational agent-based prompts to best support a given cohort of learners.

The integration of these various prompt types allows the conversational agent to provide comprehensive support throughout the collaborative programming activity. By combining role structuring, task management, and targeted reflection prompts, the agent plays a crucial role in scaffolding the learning experience and promoting effective collaboration. In the next chapter, we will see how we arrived at the final designs of these scaffolds through an iterative process of design-based research.

# Chapter 5

## Towards Measurable Learning Gains

Existing project designs used in practice are not instrumented for the level of rigor in measurement expected in educational research. This makes it difficult to determine their efficacy, let alone compare them to improvements. Furthermore, while projects are aligned with the learning objectives of the course at a high level, they are rarely defined at any level of specificity that enables measurement and design aligned to that measurement to eventually produce measurable learning gains for learners doing the projects. Absent this level of understanding and alignment, any learning gains from the project are accidental in the worst case and suboptimal in the best. Thus, in the first series of experiments in this dissertation, we undertake a rigorous redesign of the team programming project in a series of design-based research experiments.

An additional constraint that we face across these experiments is the time, effort, and level of investment required from the course instructors to make the changes we were suggesting. Consequently, updates could only be incremental, proving the value of each update before implementing it in a subsequent design. Thus, we start with a naive, lightweight implementation of the OMP paradigm in the team programming project, moving progressively to more structure and scaffolding based on the evidence gathered from experimentation. Demonstrating that rigorous instructional design and finer-grained measurement of learning are required in the pedagogy of advanced computer science courses is a contribution of this dissertation.

### 5.1 Study 1 — Uptake and Size Effects

In the first study in this series, we investigate OMP in the context of a 6-week free online Cloud Computing course offered to working professionals in the summer of 2018 [67, 136, 141]. We first see if the industry-inspired paradigm can be used to structure small group collaboration. Only then can it serve as a basis for learning from problem-solving. To that end, we investigate the following research questions —

- RQ1 — To what extent do learners follow the behavior prescribed by OMP?
- RQ2 — How does learner behavior in the OMP differ according to group size, problem difficulty, and success solving the problem in the allotted time?

As described in Chapter 4, OMP is designed to solve common collaboration issues observed in this context. RQ1 poses the question of whether that scaffold was followed in practice. RQ2 poses the question of how learner behavior changes with group size, difficulty of the problem, and whether successful and unsuccessful groups behave differently. This can help us improve the design of the OMP scaffold by determining the right-sized groups, the right problem difficulty that provides sufficiently complex learning opportunities, and even better scaffolds for stimulating productive learning behaviors while quashing unproductive ones.

### 5.1.1 Course Context and Design of the Scaffold

We investigate OMP in the context of a 6-week free online Cloud Computing course offered to working professionals in the summer of 2018. The participants had prior programming experience in Java, Python, and Bash. Importantly, participants were familiar with Mob Programming as a paradigm which had originated in the industry. Consequently, we can expect a faster uptake of the OMP scaffold and any observed differences to be ascribable to the design of the scaffold rather than to a lack of familiarity with Mob Programming. Thus, this context could serve as an intermediary for importing OMP into advanced computer science courses where learners have a similar background but much less familiarity with Mob Programming.

A series of four OMP projects was added to the course — Mob 0 through Mob 3, across four weeks of the course. Grouped by their availability, up to 6 learners were assigned to a project in order to account for those who would not show up and those who would drop out of the course entirely. Three Mob Programming roles — Driver, Navigator, and Mob member were used for the projects in this course. 3 to 6 learners in each group would be sufficient for each OMP role to be satisfied, with 1 to 4 learners in the mob role at any given time, providing space for deliberation. Each OMP project session was for the duration of a class session (80 minutes), and the roles were set to rotate every 7 minutes with a warning sent by the conversational agent 1 minute before new roles were assigned.

The OMP project was envisioned as helping learners expand on the knowledge they had gained from working on their individual projects. In other words, the individual project served as preparation for collaboration, which has plenty of precedent in the collaborative learning literature [50, 95, 106]. Consequently, each of the four OMP project sessions took place after the individual projects for that week of the course.

Special consideration was needed for the problems posed in each OMP. They needed to be difficult enough to not be solvable by an individual while allowing multiple solution paths to manifest and implementation consensus to emerge via discussion, all to lead the group to, ideally, solve the problem in 80 minutes (the duration of a synchronous project session). In other words, the problem had to be “collaboration-worthy”, a concept we elaborate on in Sankaranarayanan et al. [144].

In each OMP, a grade was assigned to incentivize participation [57], and for successfully completing the assigned problem.

Before the first project, the learners received a primer that introduced them to the OMP scaffold, its goals, the roles involved, and the responsibilities associated with each role. A walkthrough provided a step-by-step example using Test Driven Development (TDD) in the context of OMP. To put it into practice, they were introduced to the Cloud9 environment where they participated in Mob 0 around a trivial Fizz-Buzz problem <sup>1</sup>. The programming problem was kept trivial to allow the learners to focus on internalizing the group coordination aspects of OMP. Three more project sessions were conducted in the weeks following Mob 0. Each project session was held after an individual project so that the learners would have the background necessary to participate productively in the project.

Individual projects in each week of the course leading up to the OMP project sessions were as follows —

- Mob 0 — In the first week, the learners explored the offerings of each cloud service provider (Amazon Web Services, Microsoft Azure and Google Cloud Platform) used in the course. The OMP project session was an introduction to the scaffold and the Cloud9 development environment with a straightforward FizzBuzz problem.
- Mob 1 — The individual project introduced the learners to the Map-Reduce paradigm and required them to design a Map-Reduce job to filter and aggregate Wikipedia page view statistics. During the OMP project, the learners built on this knowledge by implementing more complex data processing rules in a parallelized fashion.
- Mob 2 — In the project leading up to Mob 2, the learners had to develop a software solution to automatically scale virtual machines in response to changing HTTP traffic. Subsequently, they were tasked with using cloud-based load balancers and scaling policies to maintain a given resource usage limit. The task chosen for the OMP project expanded on the load balancing concepts introduced in the individual project by having the learners implement the Token Bucket algorithm for HTTP traffic rate limiting.
- Mob 3 — In the project leading up to Mob 3, learners had to develop a video processing pipeline that integrates machine learning services and search engines to build a video indexing service. The OMP project required them to develop a new AWS Lambda function that would identify faces within thumbnail images and blur all faces in the image, using the Pillow Imaging Library.

### 5.1.2 Data Collection

The following data informed our analysis —

- Pre-Course Survey — Learners complete a survey at the beginning of the course answering questions about programming proficiency, familiarity with collaborative learning paradigms such as Pair and Mob Programming, English language proficiency, familiarity with AWS or other cloud services and their course goals.

<sup>1</sup><https://www.hackerrank.com/challenges/fizzbuzz/problem>

- Post-OMP Surveys — After each OMP project session, learners completed a post-project survey with questions about their OMP experience such as their thoughts on its impact on their learning, whether they liked the project, what worked, what did not work, and what could be done to make OMP better.
- Post-Course Survey — At the end of the course, learners completed a survey to provide feedback on the course as a whole. They also answered questions about their OMP experience, including how it will be relevant to their learning and work in the future.
- Code Revisions — Code revisions during each OMP project session were logged by timestamp.
- Chat Contributions — Chat contributions during each OMP project session were also logged by timestamp.

Table 5.1 gives an overview of the data collected in the four OMP project sessions.

### 5.1.3 Analysis and Results

#### Uptake of the Scaffold

To answer RQ1, we looked at the chat contributions and file revisions made by learners separated by the roles they assumed in the OMP project session. If the structure imposed by the scaffold was followed, we would expect to see that most of the file revisions (code contributions) come from learners in the driver role. Moreover, we would expect learners in all roles to consistently make chat contributions as an indicator of active participation in the project. Table 5.1(a) shows the number of chat contributions and file revisions by role for each OMP project session. Examining chat contributions, we see that across all the OMP project sessions, about a third came from learners assuming each of the roles. This suggests that regardless of the roles assumed, learners were not disengaged from the project. In addition, more than 80% of the file revisions in all sessions were performed by the driver, indicating that the intended role of the driver was largely followed.

Inspecting the chat contributions being made by the driver, most of them are requests for direction such as “We passed that test case, what next?” or requests for clarification such as “Are we reading from file or standard input?”. This is consistent with the role of the driver but needs further analysis to quantitatively verify, which we undertake in the next study in this sequence. Similarly, messages sent by the navigator mostly involve marshaling the mob or providing directions to the driver based on the mob’s decision. Another factor that needs to be investigated in more detail is deviations from role orthodoxy. For instance, one mob user commented in the file to organize their thoughts. These comments end up being counted as code revisions in this study. In some other cases, the role switches did not happen instantly, even with advance warning from the conversational agent. The driver would often wait to finish what they were currently working on before switching roles. Contributions made after the switch are attributed to their new roles in the current study, which could possibly explain some of

the deviation observed in the results.

Table 5.1: Number of Chat Contributions and File Revisions

(a) Chat Contributions and File Revisions by Role per OMP project session											
OMP Session	Total Users	# of Groups	Number of Chat Messages			Number of File Revisions			Driver	Mob	Navigator
			Total	Driver	Mob	Total	Driver	Mob			
mob_0	51	17	2697	866 (32.11%)	908 (33.67%)	923 (34.22%)	5178	4280 (82.66%)	395 (7.63%)	503 (9.71%)	
mob_1	28	10	1449	387 (26.71%)	595 (41.06%)	467 (32.23%)	5451	4500 (82.55%)	676 (12.40%)	275 (5.04%)	
mob_2	24	7	963	292 (30.32%)	380 (39.46%)	291 (30.22%)	1782	1530 (85.86%)	168 (9.43%)	84 (4.71%)	
mob_3	19	5	927	263 (28.37%)	358 (38.62%)	306 (33.01%)	1245	1049 (84.26%)	125 (10.04%)	71 (5.70%)	

(b) Chat Contributions and File Revisions by Role per Group Size											
Group Size	Total Users	# of Groups	Number of Chat Messages			Number of File Revisions			Driver	Mob	Navigator
			Total	Driver	Mob	Total	Driver	Mob			
3	36	12	2117	643 (30.37%)	756 (35.71%)	718 (33.92%)	5005	3759 (75.10%)	902 (18.02%)	344 (6.87%)	
4	40	10	2161	558 (25.82%)	918 (42.48%)	685 (31.70%)	3981	3560 (89.42%)	207 (5.20%)	214 (5.38%)	
5	15	3	624	170 (27.24%)	278 (44.55%)	176 (28.21%)	1639	1464 (89.32%)	106 (6.47%)	69 (4.21%)	
6	12	2	486	131 (26.95%)	252 (51.85%)	103 (21.19%)	740	549 (74.19%)	139 (18.78%)	52 (7.03%)	

(c) Chat Contributions and File Revisions by Success on the Problem (Normalized to One Learner in Mob Role)											
Problem Completion	Total Users	# of Groups	Number of Chat Messages (Normalized to One Learner in Mob Role)			Number of File Revisions (Normalized to One Learner in Mob Role)			Driver	Mob	Navigator
			Total	Driver	Mob	Total	Driver	Mob			
Success	103	32	5229	48.1 (29.43%)	61.12 (37.41%)	54.19 (33.16%)	9978	254.19 (81.52%)	32.16 (10.31%)	25.47 (8.17%)	
Failure	19	7	807	38.43 (33.33%)	40.71 (35.32%)	36.14 (31.35%)	3678	460.71 (87.68%)	47.86 (9.11%)	16.86 (3.21%)	

## Group Size Effects

The first factor in RQ2 is group size. To infer that the structure imposed by the scaffold continues to be followed as the group size increases, we would first expect the driver to make a majority of the file revisions (code contributions). We would also expect learners in all roles to actively participate in the chat, but expect learners in the mob role to account for a larger percentage of contributions because the number of members in the mob role increases with increasing group size.

To investigate this, we examine code contributions and file revisions separated by group size. Table 5.1(b) shows file revisions and code contributions by role for groups of each size. Looking at the chat contributions, we see that the learners are not disengaged in any role (at least 20% of the chat contributions came from the learners in each role regardless of the group size). We can also note that the percentage of chat contributions made by learners in the mob role increases with increasing group size from 35.71% for groups of size 3 (1 learner in mob role) to 51.85% for groups of size 6 (4 learners in mob role), which is consistent with what we would expect to find.

Looking at the file revisions, we can say that the driver role was largely followed. However, there seem to have been deviations in groups of sizes 3 and 6. These differences are not statistically significant — The number of file revisions made by drivers in a group of size 3 for example was 313.25 on average per mob (SD = 269.34), which was not significantly different from the same for groups of size 4 which was 356 (SD = 316.74) ( $p = 0.7356 > 0.05$ ). Similarly, the difference in the average file revisions per mob made by groups of size 5 which was 488 (SD = 308.95), and groups of size 6 which

was 274.5 (SD = 283.55), was also not statistically significant ( $p = 0.4934 > 0.05$ ). Even so, it might have been the case that in smaller groups, there is less social pressure to uphold rules, more instances of possible domineering by one member, or deference to one person to code. In groups of larger sizes, decisions may take longer, or the driver role takes longer to come around leading to impatience on the part of mob members, and these warrant further investigation in order to identify the ideal group sizes for Mob Programming.

### Problem Difficulty Effects

The second factor in RQ2 is problem difficulty. As problems that the groups solve become harder, we might expect that they spend more time thinking individually and less time actually discussing or making code revisions. Therefore, we would expect to see fewer chat contributions and fewer file revisions with increasing problem difficulty. The course team rated Mob 3 as the most difficult followed by Mob 1 and Mob 2. Mob 0 cannot be used even though the problem was trivial because it served as the introduction to the OMP scaffold and Cloud9 environment. Keeping this in mind, we can once again look at Table 5.1(a). While there is no clear trend, learners seem to have made fewer chat contributions and fewer code revisions on average in the case of the hardest problem, suggesting that they may have spent more time thinking individually. In-process support, such as conversational agent-based prompts can be used to encourage learners to externalize their thinking. The goal is ultimately for their mental models to be externalized, providing an opportunity for peers to build on it through discussion and, eventually, for the updated mental model to be internalized once again leading to learning. An added benefit might be that this process helps learners understand the benefits of collaborative learning — that their peers can be thought partners in helping them understand and make progress towards a solution. We investigate this form of in-process support in future studies in this series.

### Effect of Unsuccessful Problem Completion

The final factor in RQ2 is the effect of success on the problem. Understanding ways in which the behavior of unsuccessful learners differs from that of successful ones can help us better support them. In cases where the group was unsuccessful, we might expect learners to make fewer file revisions and more chat contributions if they were discussing the problem but could not come to a consensus on the path forward. We might also expect them to make more file revisions and fewer chat contributions if they were "wheel-spinning" by trying many different things instead of discussing and coming to a consensus before implementing [20]. Table 5.1c shows file revisions and chat contributions normalized to one learner in the mob role and separated by whether groups solved the problem or not. Most groups were successful. In the minority of the unsuccessful groups, learners made almost twice as many file edits and fewer chat contributions. The number of file revisions made by the learners in the driver role for unsuccessful mobs was on average 460.71 (SD = 350.47) which was marginally significantly more than the

same for successful groups which was 254.19 (SD = 220.65) ( $p = 0.1455$ ). Differences in chat contributions made by members in any role were not statistically significant. This may suggest that the students were possibly wheel-spinning. It was further observed that these students made more submissions to the auto-grader hoping for feedback that they could use to get to the right answer. This unproductive form of engaging with the support, termed "gaming the system", has been identified as one major predictor of failure [16]. Both spaces provide opportunities for improved project and scaffold designs.

## Student Perception

One goal of OMP is to provide an opportunity to jointly optimize for learning and productivity on the task. To that end, we are faced with the challenging task of shifting learner perceptions away from the current culture of fixating on grades to caring about the process and mechanisms they use for learning from their peers. To gather evidence for whether we contribute to this culture shift, and to get a deeper understanding of what learner perceptions about the project were and what they thought they gained from it, we administer surveys after each OMP project session and at the end of the course. The post-OMP surveys solicited immediate feedback on learner perceptions while the post-course survey provided an opportunity for learner to reflect on the four OMP projects as a whole and suggest possible improvements.

Three Likert scale questions asked how they liked the projects as a whole, how much they felt it improved their learning, and how their experience with Cloud9 was. The feedback was generally positive with average scores of over 3 on a 5-point scale. The answers to the open-response follow-up questions helped contextualize these scores. Collaboration and teamwork were most often cited as a part of the OMP exercise that worked best. Several learners commented about the advantage of being exposed to different approaches to the problem, having multiple pairs of eyes to catch errors and edge cases, and even commented on the benefits of collaborative programming in real time. The mob structure and rotation of roles were cited as the part of the OMP project that helped the collaboration the most. The overall sentiment is best captured by this learner's experience — *"I think working as a team and rotating roles worked best. It had people in the driver's seat to perform the actual programming and then had you sit back and see the whole picture. It also allowed us to see different approaches to problems by other people."*. However, some learners complained about their group members' lack of proficiency in programming, the mob slowing them down on things they already knew, or the speed with which the problem was being solved as a whole. This was especially pronounced in the harder projects indicating that although there were positive examples of learning, the culture shift was not entirely embraced and it is important to provide a problem at the right difficulty in order to strike a balance between productivity and learning.

### 5.1.4 Discussion

First, we situate this study in a context where learners are familiar with Mob Programming. This allows us to more cleanly interpret the results as directly pertaining to the OMP scaffold and its implementation. In general, we observe positive results in terms of the uptake of the scaffold, with the Mob Programming roles being adopted and played by the learners, as intended. Deviations from the roles are most common around the time of role switching. At the very least, learners in the driver role wait to complete the line of code or function they had been implementing before handing over control to the next driver. This suggests that a project design that allows a group configuration to remain in place until a milestone is reached may be warranted. In this proposal, however, learners may spend varying amounts of time on each milestone and therefore, in each group configuration. Worse, they may be stuck on a milestone without another group configuration to potentially break an impasse and help them make progress on the task. Careful redesign is required if milestone-based role switching is to be implemented. For now, since we see compliance at over 80%, we proceed with time-based switching.

For group size, we see the smallest and largest groups showing the most deviation, although these differences were not statistically significant. In particular, file revisions from the driver were lower in groups of sizes 3 and 6. Inspecting the chat logs, we see that groups of size 3 often let a member not in the driver role continue to implement code if they had previously signaled that they knew what they were doing. This indicates instances of domineering and deference to one person which is lessened by the social pressure seen in larger groups. At the other end of the spectrum, groups of size 6 had a similar issue that manifested differently. Members in the mob role who thought they knew the answer grew impatient of working through the scaffold to convince the group and the navigator of their idea and then for the driver to implement it. Instead, they would implement the code themselves and go as far as to submit their results to the autograder for testing. If they were successful, the group process would sometimes break down as they continued to play the role of the driver. In some other groups, they would thank the mob member for their contribution but reinforce the need to work through the scaffold because they did not understand how they arrived at the solution, or wanted to discuss other alternatives. This already shows some success in the culture shift we are hoping to create in learners — from an efficiency to a learning mindset. Nevertheless, the ideal group size might be smaller.

For problem difficulty, we observed that contributions overall were lower for the hardest problem (although not significantly so). Learners tended to contribute less when they were unsure or did not know the way forward. If unsuccessful for a long time, the groups engaged in unproductive behaviors such as repeatedly submitting slightly different solutions to the auto-tutor. Conversational agent prompt-based scaffolds would be particularly well-suited to draw out learner contributions in this case. However, this can be a form of guidance. Consequently, we reserve their addition to later studies until we have exhausted our ability to demonstrate learning from problem-solving alone, with the help of a well-designed scaffold based on minimal guidance.

Overall, the first experimental study is a promising start in producing a design that

explicitly scaffolds collaborative learning in team programming projects. The study opens many avenues of further research including group size, problem difficulty, and type, and avenues for in-process task-related support as well as support for group processes. In order to complement the above experiment, and as the first experiment in the context of an advanced computer science course, the subsequent experiment tests the OMP scaffold against student self-organization, that is, the organic emergence of a structure suited for task completion and learning. Although it is well understood in the literature that collaboration requires scaffolding, it is effortful and requires justification before instructors will implement it in their courses. The comparison is setup to explicitly gather evidence for the need — or lack thereof — for the OMP scaffold.

The one other change that was made was to move the individual project after the team programming project. Although we stated earlier that there is some evidence for the benefits of individual preparation for subsequent collaborative learning, mixed results have been reported in reality. Students were likely to encounter coordination difficulties during the subsequent collaboration [106]. For example, Tsovaltzi et al. [174] and Felton et al. [51] found that individual preparation prior to collaboration led to greater idea solidification which hindered effective group coordination, resulting in lower quality knowledge co-construction and reduced openness to alternative perspectives. The latter of the three was observed in this study, manifesting as domineering behavior from some of the learners. Thus, to provide the best change for positive collaborative behaviors to manifest, we refocus preparation before the task on declarative and procedural knowledge, distinct from what is covered in the collaborative learning task.

## 5.2 Study 2 — Comparing against Self-Organization

Reward structures used to evaluate students on their performance in team projects often incentivize behaviors like divide and conquer where students self-select into performing tasks they are already good at rather than challenging themselves to learn new skills. The OMP framework discourages the allocation of tasks purely on the basis of prior expertise, thus allowing students to not only contribute in roles they are already good at, but also learn from their teammates to contribute in roles when they are not. The first study in this series provided some evidence that this was beginning to happen. Nevertheless, this has to be compared to student self-organization to establish that these observed behaviors do not emerge organically. To that end, we hypothesize in this second study that —

- Hypothesis 1 — the OMP scaffold, if effective, will produce distinct collaborative behaviors associated with each role in Mob Programming.

If we also believe that students would default to optimizing for productivity in the absence of such a scaffold, we can hypothesize that —

- Hypothesis 2 — these distinct collaborative behaviors will not be adopted in self-organized groups, which could result in student behavior looking far more con-

sistent throughout the project.

However, by enforcing the OMP scaffold for collaboration, we run the risk that productivity may be harmed because students with less expertise at each task may get in the way. Furthermore, the cognitive load from role-switching might reduce productivity on the task, and putting students in roles they are not familiar with could increase discomfort and, therefore, negatively affect their perception of the task. Consistent with the Student Perception analyses in the prior study in the sequence, therefore, we hypothesize that —

- Hypothesis 3 — students from the OMP scaffold groups may feel more negatively about their experience compared to students in the self-organized groups and could perform worse on their project.

### 5.2.1 Course Context and Experimental Design

We experimentally contrast the OMP scaffold against student self-organization in a between-subjects design embedded in the Cloud Computing Course described in Chapter 3 [175]. A total of 120 students took the course and organized themselves into teams of 3 for a half-semester-long course project.

In the first week of the course, as part of the primers that the students had to complete, they were randomly grouped into teams based only on their availability to participate in an OMP training session. Prior to this training session, students received materials explaining the Mob Programming paradigm and how it will be used for collaborative programming exercises in the class. The training session itself required students to solve a relatively simple programming task collaboratively. As before, the task was kept simple to allow students to familiarize themselves with the Mob Programming paradigm as well as the Cloud9 interface, which was the software development platform that housed their collaborative project, as described in Chapter 4. Each project session lasted 80 minutes with roles switching every 8 minutes. The role-switching was kept relatively frequent in order to promote observation of the problem from multiple perspectives.

The experimental manipulation took place two weeks after the OMP training session, when the students had acquired the prerequisites necessary to complete the programming task. As in the training session, the project session lasted a total of 80 minutes with switches occurring every 8 minutes. The students in their 3-person teams decided on a time when they would be available to work on the project together and informed the course instructors. The programming task required the students to perform data analysis on Twitter data using tools in the cloud. Many possible solution paths and ways of implementing the solution provided room for the responsibilities associated with each role to manifest in concrete ways towards producing the end product. Students in both conditions performed their collaborative work during the manipulation in the same Cloud9 environment. However, only in the OMP condition did the students receive instructions from the conversational agent to engage in OMP practices and experience automated support for role-taking and timed role switching.

## 5.2.2 Data Collection

The following data was collected in both conditions to facilitate our analyses —

- Code contributions and chat logs from the team programming exercise — These allow us to observe how students organized themselves in the self-organized condition and see if they followed the structure imposed by the scaffold in the OMP condition.
- Grades — Grades on individual assignments and the team project before and after the team programming exercise help control for differences in prior knowledge among students assigned to either condition. The grades on the team programming project itself help us see if there was a difference in group product quality across conditions.
- Post team programming project survey — This survey asked about prior familiarity with teammates, how this project helped discover teammates, how they chose to structure the project, how their experience with the OMP training session helped structure this project, how effective they felt their organization was, both in the training session and in this project, and their experience with the Cloud9 interface. The survey helped us answer the question about student perception, but also allowed us to better understand the mechanisms behind the observed results.

## 5.2.3 Analysis and Results

### Hypothesis 1

The OMP scaffold, if effective, will produce distinct collaborative behaviors associated with each role in Mob Programming.

One goal of OMP is to orchestrate the rotation of team members through a set of three distinct but interdependent roles. If the manipulation was successful, we would expect to see distinctive behavior patterns associated with the roles, in support of Hypothesis 1. As both a manipulation check and a lens for elucidating the effect of the manipulation on collaborative processes, we conducted a quantitative discourse analysis.

In order to test the distinctiveness of the roles that team members took up in the discussions, we identified themes within the discussion using an automated technique. First, we aggregated the transcripts for all discussions so that each turn was labeled with a condition and a role. We removed the contributions from the conversational agent. In total, there were 3,396 remaining turns produced by 36 groups and a total of 108 students. There are several topic analysis techniques for identifying themes automatically within textual data; however, many of them require very large text corpora in order to operate properly. Thus, we adopted a very simple technique based on the same principles. In particular, we first computed a term frequency by document frequency matrix where each document was the text uttered within a conversational turn, and its representation within its row within the matrix contained a count for each word (term), signifying how many times it appeared in the corresponding document. Then using a

Principal Components Analysis with Varimax Rotation, we identified the top 15 latent factors that distinguished conversational turns that explained the most variation in the matrix. We then saved the loading onto the 15 latent factors in the data table so that each turn was represented in terms of the 15 factors.

We selected 6 latent factors to interpret, shown in Table 5.2, which were the most distinguishing between roles in the OMP condition and behaviors in the unsupported condition based on a Chi-squared analysis between the factor loadings per turn and the turn status, which was the role the speaker was in when uttering the turn (Supported Driver, Supported Navigator, Supported Mob, or Unsupported Mob).

Factor	Example Keyword	Example Turn
Reporting Structure and Interpretation	Type, says, explain, extra, where, info	"I just did but the time doesn't improve a lot"
Evaluation and Brain-storming	away, optimized, range, tables, database	"To fix your problem, you have to select from your database"
Direction	You, near	"You can use 'submitter' to submit your solution."
Code	Crime_id, date_time, limit	"SELECT crime_id from service s where s.report_date = (select max(report_date) as date_time"
Code Status	line, at, error	"there is a syntax error with LIMIT 1"
Code Abstraction	environment, execute, line, mysql	"Let me execute this on mysql directly"

Table 5.2: The 6 most distinguishing factors with their keywords and example turns as identified by PCA

Next, we conducted a student-t post-hoc analysis to determine which comparisons between turn status were significant. If Hypothesis 1 is supported, we would see turn status between roles to be more distinctive in the OMP condition, meaning different roles are showing different patterns (factors) in the way they talk.

Here we see that turns from Drivers were distinctive in their relatively high loading on the Reporting Structure and Interpretation factor. The turns from Navigators were distinctive in terms of their relatively high loading both on Direction and Code. The turns from Mob members in the supported condition were high in Evaluation and Brainstorming and Code. However, members' turns in the unsupported condition did not load very high on any of the distinguishing factors. The distinctions we see in the interpretation of the factors that distinguish between supported roles are consistent with what we would expect based on their definition.

Thus, we do see different roles in the OMP condition adopting different discourse

	Driver	Navigator	Supported Mob	Un-supported Mob	Comparison (Student-t)
Reporting Structure and Interpretation	0.14	-0.07	0.06	-0.03	D > N
Evaluation and Brainstorming	-0.09	0.02	0.14	-0.02	SM > D, UM
Direction	-0.01	0.14	-0.03	-0.02	N > D, SM
Code	-0.06	0.07	0.11	-0.01	N, SM > D
Code Status	-0.06	-0.04	-0.04	0.03	n.s.
Code Abstraction	0.08	-0.08	-0.05	0.01	n.s.

Table 5.3: Comparison of factor loadings from PCA associated with each role, focusing on the 6 identified themes

patterns, with Driver displaying more reporting, structure and interpretation, Navigator displaying more direction, and the mob displaying more on evaluation and code. This is consistent with our definition of the three roles and Hypothesis 1 is thus supported.

## Hypothesis 2

Hypothesis 2 — these distinct collaborative behaviors will not be adopted in self-organized groups, which could result in student behavior looking far more consistent throughout the project.

We considered that it is possible that within self-organized groups members took on roles despite not having been assigned. If this was the case, we might not see those distinctively in the analysis above since turns from all team members are taken together in the self-organized condition as Unsupported Mob, and thus we would only be able to see average behavior across roles (if any). In order to test Hypothesis 2, therefore, we had to adopt a different methodology — cluster analysis. We clustered turns using K-means clustering in order to identify cross-cutting factor profiles across turn statuses. Since there are 4 Turn Statuses (Supported Driver, Supported Navigator, Supported Mob, and Unsupported Mob), we set the number of clusters to 4. In order to test whether the cluster analysis identified patterns that distinguished some subset of unsupported mob turns from those of the supported roles, we conducted a Chi-Squared test between the cluster assignment per turn and the Turn Status, however, this was not significant. One cluster contained only turns from the 3 supported roles, and in particular contained all turns from the Driver role. However, none of the clusters were distinguishing for the unsupported condition in contrast to the other supported roles.

In particular, the other clusters contained a mixture of unsupported turns, the Navigator and the supported Mob.

This cluster analysis shows that we are unable to identify any distinctive behavior patterns for turns within the self-organized groups, suggesting that the collaborative behaviors associated with each role in Mob Programming were not naturally adopted by students if not explicitly scaffolded. Hypothesis 2 is thus supported.

### Hypothesis 3

Hypothesis 3 — students from the OMP scaffold groups may feel more negatively about their experience compared to students in the self-organized groups and could perform worse on their project.

In the post-programming exercise survey, we asked students about their experience with the Cloud 9 environment and how prepared they were for future group projects. We compared student responses in the two conditions in order to test Hypothesis 3. The outcome measures used were student responses to the Likert scale questions – 1) “How prepared do you think your team is to start working on the project? (5 meaning very prepared) and 2) “What was your experience with Cloud9?” (5 meaning very positive experience). We also operationalized covariates to control for students’ prior familiarity with group members and prior knowledge. Prior knowledge was operationalized as the average score of the 2 individual exercises that the students had completed prior to this project. Familiarity was the average of the response to 2 Likert scale questions asking students to rate their familiarity with their teammates (5 meaning very familiar). We then built linear regression models to investigate students’ self-reported level of preparedness and experience with Cloud 9 across the OMP scaffold and unsupported conditions. We used the Likert-scale response as the outcome variable, condition as the main factor, and students’ prior grades and familiarity with the team as covariates. Students in the OMP scaffold condition perceived themselves as less prepared to start working on the project compared with students in the unsupported condition ( $p = 0.0413$ ). Students in the OMP scaffold condition were also less satisfied with Cloud 9 compared with students in the unsupported condition ( $p = 0.0627$ ). The effect is marginally significant. Importantly however, a linear regression model built using the mob session grade as the outcome variable, and the condition as the main factor while controlling for students’ prior grades and familiarity with their team members found no significant differences between the two conditions suggesting that while students provided with the scaffold experienced marginally significantly more discomfort, this did not manifest itself in actual performance differences. We discuss below some reasons why we think this might be the case. Hypothesis 3 is thus partly supported.

#### 5.2.4 Discussion

In our analysis, we find full or partial support for all three hypotheses. Although it might seem trivial that students in the OMP condition would take on behavior patterns consistent with the roles we instructed and scaffolded during the project, and students

in the self-organized condition would not, it is notable that just two weeks earlier, all of the students went through an OMP training session in which they were instructed about the roles and experienced the scaffolding. As a validation that students benefited from that experience and sought to apply it with or without instruction to do so, we included questions in the post-team exercise survey asking students to report whether they felt the training session before this programming exercise was. As reported in the survey, students, in general, felt the training session influenced their behaviors in the programming exercise in both conditions (with an average of 3.55, and a standard deviation of 1 on the Likert scale from the survey question — “Earlier in the semester, you participated in a training session for Online Mob Programming (OMP). Do you feel that experience influenced your behavior during this team exercise?”). The self-organized groups reported higher influence than the OMP scaffold groups, with an average of 3.72 (SD = 0.96) for the self-organized groups and an average of 3.43 (SD = 1.01) for the OMP scaffold groups. The difference between the two conditions is not statistically significant. This result validates the need for scaffolding to reliably, consistently and sustainably enable collaborative behaviors to manifest.

We also saw evidence that students felt more organized in the team programming exercise compared to the first training session regardless of the condition they were in. Two Likert-scale questions asking — 1) “How effectively did the organization work between participants during the training session?” and 2) “How effectively did the organization work between participants during this exercise? (5 meaning very effectively organized)” showed increases. Specifically, for self-organized groups, the self-reported organization measure went from an average of 3.55 (SD = 1.12) in the training session to an average of 4.24 (SD = 0.83) in the team programming exercise. For the OMP scaffolded groups, the self-reported organization measure went from an average of 3.49 (SD = 1.07) in the training session to an average of 4.08 (SD = 0.80) in the programming exercise session. There was no statistical difference between the two conditions on the organization measure. This suggests that students start to become more proficient in their roles with practice and shows promise for reducing the discomfort associated with the scaffold, as well as the increased possibility of producing beneficial learning opportunities over time. There could have been several reasons why students felt discomfort with the OMP scaffold. An open-ended question in the survey asking “What could be done to improve the exercise?” points to a few possible explanations. Several students were disgruntled that the roles were not assigned separately by task or their ability to do those tasks — “the roles can be changed according to tasks and individual’s ability”. This supports our hypothesis that students are focused on productivity rather than using the task as a learning opportunity and require a scaffold to make the best use of the project for learning. In a similar vein, several students thought that a voice communication channel would produce more efficient and quicker communication, reinforcing once again the fixation on productivity. A third possibility is discomfort with the Cloud9 IDE that several students expressed needing more practice with. As we observed above, this form of discomfort is likely to decrease with practice. Finally, as in the first study, we observed some instances of domineering and deference from group members where individuals broke from their suggested roles. As in the first study, this could suggest

that a slightly larger group size may be warranted to increase social pressure while also not introducing the problems observed with even larger groups. We settle on groups of size 4 based on the results of these two studies.

Although we do not yet measure learning from the task explicitly, we did conduct an analysis to investigate whether the OMP scaffold session had any impact on students' grades later in the course. We computed the post-OMP project session grade by averaging the four assignment grades that occurred after the OMP project session and refer to it as post-OMP grade. We used post-OMP grade as the outcome variable, condition as the main factor, and controlled for students' prior grades and familiarity with the team. We do not see a difference in post-OMP grade between the two conditions. This suggests that in the OMP scaffold session, students picked up the distinctive collaborative behaviors associated with each role, while at least not harming their individual learning to prepare them for the later stages of the course. Exposure to more OMP project sessions could help us see decreases in the initial discomfort associated with the adoption of the scaffold and start to see more of the learning gains.

### 5.3 Study 3 — Added Implicit Support and Comparison with Individual Projects

Having reasonably settled questions about the adoption of behaviors scaffolded by the OMP paradigm, we can start to turn our focus towards designing for learning from the project. This study seeks to evaluate the use of implicit and explicit scaffolds for collaborative learning in tandem. The OMP scaffold is an explicit one. In contrast, in prior research, we had shown that intelligently forming teams based on maximizing measures of transactive exchange from prior student interactions acted as an implicit scaffold for collaboration [137, 138, 196, 198]. Transactive exchange refers to a pair of utterances in which the second utterance operates on and builds on the reasoning of the first [22]. They can, of course, be "self-oriented", where an individual is building on their own reasoning. However, "other-oriented" transacts, or individuals operating on the reasoning of others, are more interesting from the standpoint of collaborative learning. They can lead to the emergence of transactive memory systems, a proxy for collaboration effectiveness [35]. The hypothesis validated in prior research was that demonstration of collaboration effectiveness, measured by a reasonable proxy, can be used to form teams for future collaboration where this collaboration effectiveness continues and is better than in teams formed randomly. Implicit forms of scaffolding have some precedence in the literature from the standpoint of efficacy [188]. The more common motivation to attempt implicit light-touch forms of scaffolding is to evaluate if explicit scaffolding is warranted and only then design for it. Prior research using the same implicit scaffold had shown that combining the two could produce synergistic effects [190], however, which prompted our design in this study. In a 3x3 Latin square design, we compare students working alone and in two OMP configurations (with and without transactivity-maximizing team formation) [143].

Because the OMP paradigm separates the responsibilities into three distinct roles associated with brainstorming ideas, selecting among the ideas being considered by analyzing their pros and cons, and implementing the selected idea, we can expect that the more teams adhere to the OMP paradigm, the more substantive their discussions will be. The distribution of responsibilities to roles in an interdependent manner can also ameliorate group coordination difficulties and process losses, thus ensuring that the quality of the group product does not decrease. Moreover, the rotation of roles affords a more even distribution of responsibilities among all of the participants. We can hypothesize, therefore, that —

- Hypothesis 1 — Teams that demonstrate an elevated level of compliance with the OMP paradigm will discuss conceptual content relevant to the project more substantively, contribute work towards the group solution more evenly, and produce a group product of as high quality as individuals or teams with lower compliance to the OMP paradigm.

In designing this current study, we build on previous work in developing a team formation strategy that is associated with increased idea sharing [198] and reduced problems with labor distribution and conflict [138]. The benefits of transactive idea exchange can potentially interface well with the hypothesized benefits of OMP including more even distribution of work and more substantive discussions. Thus, we hypothesize that the assignment of teams using this transactivity maximization approach might increase adherence to OMP practices and further amplify its benefits.

- Hypothesis 2 — Groups formed transactively will demonstrate higher compliance with OMP practices that will be associated with an intensification of the observed benefits of OMP compliance.

### 5.3.1 Course Context and Experimental Design

In order to test the two hypotheses specified above, we experimentally contrast the OMP scaffold in randomly formed and transactively formed groups against individual programming in a 3x3 Latin square between-subjects design. As before, the experiment is conducted in the online Cloud Computing course offered to the students of Carnegie Mellon University and its campuses worldwide. The comparison between individual programmers and groups will allow us to investigate Hypothesis 1 and the comparison between transactively formed and random groups will allow us to investigate Hypothesis 2.

A total of 120 students took the course, allowing 40 students to be assigned to each condition for each exercise. In the first week of the course, as part of the primers that the students had to complete, they were randomly grouped based only on their time availability to participate in an OMP training session. Prior to this training session, students were provided with materials explaining the Mob Programming paradigm. The training session itself required students to solve a relatively simple programming task collaboratively. The task was kept simple to allow students to familiarize themselves with the Mob Programming paradigm as well as the Cloud9 interface. Each project

lasted 80 minutes with 10 minutes reserved for introductions and wrap-up and roles switching every 7 minutes. The role-switching was kept relatively frequent in order to promote observation of the problem from multiple perspectives. All students first took part in this OMP training session four weeks before the start of the experimental manipulation, thus allowing them to familiarize themselves with the workings of the OMP framework and the Cloud9 interface.

The experimental study was carried out as a Latin square design of 3 (condition)  $\times$  3 (programming project), where the order of the conditions was counterbalanced between three tracks to control for potential order effects. The three conditions were Individual, Random Team Assignment, and Transactivity-Based Team Assignment. The students were randomly assigned to the three tracks. Each time team formation was performed, it was done within a track such that all students within that track on that assignment were assigned to teams using the same paradigm. Teams of 4 students were formed. Since within each track, team assignment was performed for two different programming projects, we ensured that teams were formed with students who had not worked together at the other time points, including the training that occurred prior to the experimental manipulation.

### The Reflection-Feedback Exercise

The transactivity-maximizing groups were formed by maximizing the pairwise transactive exchange observed from a Reflection-Feedback discussion forum exercise conducted prior to the experimental manipulation. This exercise is described below.

In the initial part of the semester, the students worked through individual projects. After each project, they were required to post a reflection to the discussion forum. Some examples of questions posed to the students are shown below —

- Pick a task you found most challenging. Why was it challenging and how did you end up solving it?
- Pick a task and choose among different solutions paths for this task. What were the trade-offs you ended up making?
- Describe how you tested one of the tasks. How did you design your test? Was your initial test sufficient? If not, how did you improve it?

After responding to these reflection questions, their answers were shared on a discussion forum that the entire class could access. Students were then encouraged to provide constructive feedback to at least three other reflection posts. An example reflection, prompt, and feedback post can be seen in Fig. 5.1.

Substantive discussions resulted from this reflection-feedback exercise as shown in the example feedback post. These posts showed evidence that students synthesized the knowledge from several posts, gathered together, and provided encouragement to each other. An example of a transactive and non-transactive exchange between students from this reflection-feedback exercise is shown below —

- **Transactive Exchange**

*Student 1: "...I used 'f.readlines()' to read the wiki log file. It worked well on my own*

 @andrew.cmu.edu 10 Sep

I really spent a lot of time in this project(more than 40 hours). In the first 15 hours, I was all considering the workflow of this project. I read all the relative primers of this project, and I even created a S3 bucket and installed AWS SDK and AWS eclipse plug-in. Most of the things I tried proved to be no use. Wating much time here really reflects my lack of background knowledge. When I do the filter job, the most difficult part is how to keep it encoding in UTF-8 every where. I have few experiences programming in multiple environments. It really cost much time for me to find a way to make it consist in UTF-8. For the questions, to be honest, I have never used awk and grep before, and I am even not familiar with the Linux basic commands. I finished those questions while learning from them, and I found awk and grep very powerful. I was mainly using JAVA in the project, while in q7 I used Python. Pandas is really an amazing lib to solve such problems, so I chose to switch to Python even I almost have no previous Python programming experience. It works very well, and I felt it crazy that the problem is solved in only few lines. Another thing I felt interesting is the regex. I spent a long time working on it, especially in q8. Previously I have seen someone use things like ".\*" to represent zero or more occurrences of any characters, and now I know how did they come up with such a rule.

 report  share  like (0)  quote  reply

---

 CloudComputingBot 10 Sep

Please provide constructive feedback to your peer's post and substantiate your criticism/suggestions with reasons. Evaluate the arguments presented, be clear with your arguments and use supporting evidence.

 report  share  like (0)  quote  reply

---

 @andrew.cmu.edu 11 Sep

My situation is a little similar to yours, but don't worry about spending lots of time on primers. I think you'll save a lot of time in the following projects because of doing this. I've never used python before, either. Besides, according to classmates' summaries of this project, I think python is really efficient when solving certain problems compared to java. Congratulations that you have learned to use it in q7!

 report  share  like (0)  quote  reply

Figure 5.1: Examples of a reflection post, a prompt soliciting constructive feedback and a feedback post. Feedback post highlights instances of common ground, synthesis and encouragement.

*computer, but it caused a 'MemoryError' when I tested it on AWS ... "*

*Student 2: "The file object itself is a iterator. So if you 'for x in file', you get lines as x. This is a more pythonic way than using 'readline()'."*

- **Non-Transactive Exchange**

*Student 1: "...I approached the problem by breaking it out into different modules and functions which made it possible to test different cases really fast. ..."*

*Student 2: "Well done!"*

In the first case, the second student is explicitly referring to the reasoning of the first student and building on that reasoning further with their own reasoning. The interaction is therefore transactive. In the second case, the second student is referring to the reasoning of the first student but is not contributing original reasoning of their own, and the interaction is therefore non-transactive. Thus, evidence of transactive exchanges can be mined from these interactions to automatically inform our team formation algorithm.

This feedback exchange provided the opportunity for students to experience more social interaction in the course. At the same time, we can use the data to estimate collaboration potential for pairs of students based on their exchange of transactive feedback contributions. An automated measure of transactive exchange between students in this context is first used to estimate pairwise collaboration potential, and then a constraint satisfaction algorithm is used to assign teams in such a way that students are more likely to be part of teams with the other students with whom they have interacted transactively than those with whom they have not interacted transactively.

## Automatic Transactivity Analysis and Team Assignment

Before an estimate of pairwise transactivity exchange can be computed, posts in the discussion forum must first be annotated as transactive or not. In our work, this was accomplished using an automated text classification approach developed in prior work on automated collaborative learning process analysis [6, 75, 197]. This approach requires training data including a validated and reliable coding of transactivity [62]. For our work, we used a previously validated coding manual [62] and coded 200 feedback exchanges by hand. Using this training data, we trained a model to perform the transactivity analysis over the entire set automatically.

For each pair of students, we computed the total number of threads where either they both contributed a transactive post to the discussion or one of them started the thread and the other contributed a transactive post. We refer to this quantity henceforth as the Pairwise Transactivity Score for a given pair of students. Once the Pairwise Transactivity Score is computed for each pair of students, a team score can be computed by averaging the Pairwise Transactivity Score for each pair within the group. A score for the resulting teams across the whole class can be computed by averaging across the team scores. The goal of the automated team matching algorithm is to assign students to teams in such a way that the score is maximized throughout the class. An exhaustive search would take inordinately long. Thus, a constraint satisfaction algorithm is used to find an approximate solution that comes close to the optimal assignment that

maximizes the score in the class without having to perform an exhaustive search. The specific constraint satisfaction algorithm we used is called the minimal cost max network flow constraint satisfaction algorithm [5]. The algorithm generally tackles the resource allocation problem with constraints, and in previous work, role assignments such as the roles of a Jigsaw condition were used as constraints [197]. In this paper, location was used as the constraint in addition to maximizing average transactivity across teams, that is, all members of the team are co-located on the same CMU campus. We covered the reasons for this additional constraint while describing the context in Chapter 3. The algorithm finds an optimal grouping within  $O(N^3)$  time complexity where  $N$  is the number of students. A brute-force approach would have  $O(N!)$  time complexity and would be infeasible in practice.

The algorithm is capable of forming teams of arbitrary size and approximates the solution in admissible time by maximizing the transactivity post-count between two adjacent pairs of users instead of the total accumulated transactivity post-count. A discussion network which is a directed weighted graph of the student's discussion in the reflection-feedback phase weighted by the transactivity score is built and the successive shortest paths algorithm shown in Algorithm 1 greedily finds the minimum-cost flow until there is no remaining flow in the network. The algorithm can be extended to

---

**Algorithm 1** Successive Shortest Paths for Minimum Cost Max Flow

---

```

1:  $f(v_1, v_2) \leftarrow 0 \forall (v_1, v_2) \in E$ 
2:  $E' \leftarrow a(v_1, v_2) \forall (v_1, v_2) \in E$ 
3: while  $\exists \Pi \in G' = (V, E')$  s.t.  $\Pi$ , a minimum cost path from source to destination
   do
4:   for each  $(v_1, v_2) \in \Pi$  do
5:     if  $f(v_1, v_2) > 0$  then
6:        $f(v_1, v_2) \leftarrow 0$ 
7:       remove  $-a(v_2, v_1)$  from  $E'$ 
8:       add  $a(v_1, v_2)$  to  $E'$ 
9:     else
10:       $f(v_1, v_2) \leftarrow 1$ 
11:      remove  $a(v_1, v_2)$  from  $E'$ 
12:      add  $-a(v_2, v_1)$  to  $E'$ 
13:    end if
14:  end for
15: end while

```

---

accommodate more than one constraint, but it should be noted that adding additional constraints could mean that an optimal team assignment ceases to exist.

## Experimental Manipulation

The first project of the experimental manipulation started four weeks after the OMP training session, and the other two projects were administered in two-week intervals

after that. As in the training session, each project lasted a total of 80 minutes with 10 minutes reserved for introductions and wrap-up and role switches prompted by the intelligent conversational agent facilitator happening every 7 minutes. In the first project, students worked on the concept of thread synchronization to simulate a bank system that handles deposits and withdrawals; in the second project, students used functional programming in Scala to implement a simple application using Spark Distributed Databases; in the third project, students solved a binary classification problem by extracting and engineering numeric and categorical features in Python. Students in all 3 conditions performed their work during the manipulation in the same Cloud9 environment. They were supported by the AI conversational agent that provided instructions, managed the time for the project, and informed the students of their roles and when they needed to change roles.

### 5.3.2 Data Collection

The following data was collected in all conditions to facilitate our analysis —

- Code contributions and chat logs — Code contributions allow us to analyze how work was distributed among the group members. Because the student in the Driver role is required to do all of the code changes, we can also measure if students complied with the structure of OMP or not, using the code contributions. Chat logs allow us to analyze the content of discussions between students.
- Grades — Grades on individual assignments and the team project before and after the OMP exercises help control for differences in prior knowledge and ability among students assigned to any condition. The grades on the exercises themselves, which help us see if there was a difference in group product quality across conditions.
- Post-OMP and post-course surveys — In addition to asking for feedback, the surveys were used to gather information on student perception of the project, the learning from the project and the group product. The survey provides additional context for analyzing the quantitative data obtained from the logs.

To quantify whether the students complied with the structure suggested by the OMP project, we calculated a Compliance Score for each of the groups. The compliance score was measured as

$$\text{compliance\_score} = \text{code\_driver} / \left( \frac{\text{code\_not\_driver}}{n - 1} \right)$$

where **code\_driver** is the number of edits to the code made by participants in the driver role, **code\_not\_driver** is the number of edits to the code made by everyone else and **n** is the number of group members. A higher compliance score, therefore, means that the ratio of the code contributions made by the driver to the number of contributions made by the rest of the members is more, and that constitutes more compliance with the OMP structure.

In order to further pin down how work was being distributed in the group, we calculated the percentage of code contributions made by each group member. From this, we computed an Evenness Deviation score, which measured the difference between this percentage and what percentage would be observed for the group if work was distributed evenly. This number was scaled in order to allow for values to vary between 0 and 1 regardless of group size.

Finally, in order to quantify the extent to which project-relevant Conceptual Content was being discussed in the chat, we measured the vector similarity of the topic representation of the chat contributions of a student with that of the primer corresponding to that project. The topic representation was constructed using a latent semantic indexing model over a bag-of-words representation of the set of primer documents using the number of topics set to 5. A higher document similarity score meant that more of the conceptual content from the primer was discussed in the chat.

### 5.3.3 Analysis and Results

The formulation of our hypotheses and the accompanying study design was based on the concern that when students are assigned team work in a project course, dysfunctional teams function like the most capable student within the team because that person takes on the lion's share of the work, thus undercutting the practice opportunities of teammates. So, as a foundation for the evaluation of the hypotheses, we first quantified the extent to which this problem was in evidence within the course we chose to study.

We begin by checking to ensure that students in all three conditions achieved equivalent grades on the projects and that there were no significant differences. For this test, we computed an ANCOVA model with Condition at a time point as the Independent variable, average grade prior to the experiment as Covariate, Project time point as a random variable, and Grade at time point as the Dependent variable. There was no significant effect of Condition either for the auto-graded portion of the projects ( $F(2,348) = 0.17$ ,  $p = 0.88$ ) or for the manually graded portion of the projects ( $F(2,348) = 0.83$ ,  $p = 0.43$ ). Thus, teams (regardless of condition) and individuals achieved the same grade on average. For the remainder of the analysis, we will focus only on teams.

We have observed students who default to an uneven distribution of labor in order to achieve an advantage on their grade. In order to test the extent to which the reward structure does encourage an uneven distribution of labor, we computed an ANCOVA model with a three-way split on the Evenness Deviation variable (Top Quartile, Middle, Lower Quartile) as the independent variable, average grade prior to the experiment as Covariate, Project time point as a random variable, and Grade at time point as the Dependent variable. The upper quartile had deviation scores of higher than 0.33, and the lower quartile had deviation scores less than 0.08. The median deviation score was 0.2. In 3 percent of teams, a single member did all of the work. There was no significant effect of the deviation variable on grade, though the trend was in the expected direction both for the auto-graded and manually graded portions of the project. Thus, students may falsely believe that it is necessary to deviate from an even distribution of labor when, in fact, it does not help their grade.

## Hypothesis 1

- Hypothesis 1 — Teams that demonstrate an elevated level of compliance with the OMP paradigm will discuss conceptual content relevant to the project more substantively, contribute work towards the group solution more evenly, and produce a group product of as high quality as individuals or teams with lower compliance to the OMP paradigm.

We first tested for an association between Compliance scores with Conceptual Content scores and found the association to be highly significant ( $R=0.2$ ,  $p <0.005$ ) such that more highly compliant groups focused more on conceptual content in their chats. Then we compared the Conceptual Content scores for students in compliant teams with those in noncompliant teams. We computed a median split on the Compliance score in order to compare students in groups that were highly compliant with OMP practices versus those in groups that were not. Then we compared Evenness Deviation scores for students in compliant teams with those in noncompliant teams using a t-test and found that Evenness Deviation was significantly higher in noncompliant teams  $F(1, 161) = 4.2$ ,  $p <0.05$ , effect size 0.3 SD. Just as we tested the effect of condition on grade, we also tested compliance to Mob practices. For this test, we again computed an ANCOVA model with the median split on Compliance at a time point as the Independent variable, average grade prior to the experiment as Covariate, Project time point as a random variable, and Grade at time point as the Dependent variable. There was no significant effect of Compliance. Thus, we have correlational, though not causal, evidence to support the hypothesis that Mob practices are associated with a more conceptual focus and more even distribution of labor.

## Hypothesis 2

- Hypothesis 2 — Groups formed transactionally will demonstrate higher compliance with OMP practices that will be associated with an intensification of the observed benefits of OMP compliance.

To test Hypothesis 2, we computed an ANOVA model with Condition as the independent variable and Project time point as a random variable. Compliance score was the dependent variable. Here, we found a trend consistent with the hypothesis, but it was not significant. Thus, Hypothesis 2 is not supported. One explanation is that the OMP structure acts as its own scaffold for idea exchange, which might also make the transactionality manipulation less necessary. This might explain the lack of significant support for Hypothesis 2. Considering this, future iterations drop the implicit support scaffolding.

### 5.3.4 Discussion

First, groups of size 4 were adopted for this study after evidence from two prior experiments showed that too large as well as too small groups were detrimental to compliance to the OMP scaffold. While the tendency was lesser than observed in prior studies, compliance to the roles was still not perfect and did not differ significantly based on whether

teams were formed on prior evidence of transactive exchange or randomly. Before we choose to impose compliance, however, we ask the question of whether that is beneficial.

The results show a more even distribution of labor and an increased conceptual focus in the discussions for groups with higher compliance. In other words, off-task discussions were lesser in groups that complied more with the OMP scaffold. Deviating from the scaffold was not even necessary for improved performance, since there were no significant difference in project scores even with the individual project condition. Thus, we are starting to see benefits of compliance to conceptual learning from the OMP scaffold, which we should measure with more sensitive assessments rather than relying on post-hoc discourse analysis. Diving deeper into non-compliance, we see that while the instances are fewer than in prior studies, the reasons are largely the same. That students are not task switching when told to and the only clean switch is when a milestone is reached. Therefore, milestone-based switching could indeed be warranted, but with changes to the project design such that a disproportionate amount of time is not spent on a single task within the project and students get to experience all of the Mob Programming roles over the course of the project. Taken together, this suggests a few design changes for the subsequent study.

First, we require more sensitive measures of conceptual learning from the task, which is, at the moment, a correlation, observed from the OMP project. Second, we need to redesign the project to support milestone-based switching rather than time-based switching to improve compliance. Third, we do not need additional implicit forms of scaffolding since the OMP scaffold appears to sufficiently support manifesting distinct collaborative behaviors in students. Finally, to help conceptual learning from the task translate into performance on the individual project, we may require at least some guidance in the form of conversational agent-based prompts. In particular, the prompts can be used to draw out reasoning from students and require others to build on already expressed reasoning. This addresses issues such as wheel-spinning, and encourages unblocking progress through discussion when groups reach an impasse. As an explicit, targeted form of scaffolding, it can also help us better direct students towards the learning objectives of the project. Of course, that carries a secondary implication —that the learning objectives themselves have to be articulated at a finer grain. As is, the learning objectives are only described at the level of the project which reduces the ability to design targeted support. Better defining and communicating learning objectives could also improve of changes of students realizing measurable learning gains from the project. This is supported by analogous research in subgoal labeling [100, 101]. We investigate the impact of this full redesign along with added guidance from conversational agent-based prompts in the subsequent study.

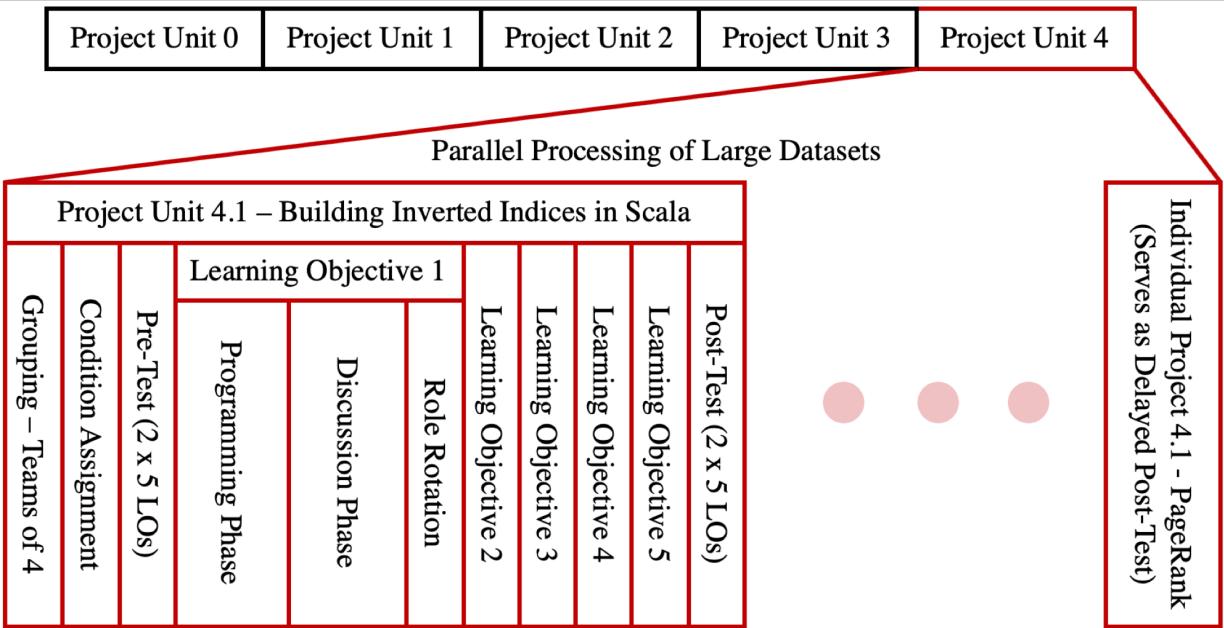


Figure 5.2: Course structure, pre-test, post-test, and delayed post-test alignment

## 5.4 Study 4 — Design Overhaul and Conversational Agent-Based Prompts

### 5.4.1 Course Context and Experimental Design

One major design change in this iteration is the introduction of pre- and post-test learning measures. An average of grades from before and after the project was used in the previous studies. While this may have shown that students were coming into the project with approximately the same levels of prior knowledge, it was not fine-grained enough to help us measure immediate learning benefits from the project itself. Further, as is typical with university courses, achievement is measured with performance on the task, but whether students are learning during the bulk of the process is not answered. In this study, we raise that question and identify that only some portions of the instructor-designed project appear to produce learning as measured by pre- to post-test learning gains [144, 145].

A summary of the course structure and the location of the study within it is shown in Figure 5.2. Within the first sub-unit of the fourth project unit of the course, students work with our collaborative software development project, now called the Online Programming Exercise (OPE), to build an inverted index using the Scala programming language. A total of 101 students from three campuses completed the task, and 100 of these students completed the subsequent project.

As described in Chapter 3, we conduct a learning science and instructional design based overhaul to the task design. We divide the overall programming project into five different tasks which target five granular learning objectives (LOs). Each LO is assigned

two multiple-choice questions each on the pre- and post-tests to measure student learning from the project. Student performance on the subsequent individual project associated with the task then serves as a delayed post-test. Table 5.4 shows the LOs and examples of pre- and post-test questions corresponding to each task, and Figure 5.2 shows the position of pre-, post- and delayed post-tests within the course.

#	Learning Objective	Example Pre/Post Test Question	Information Prompts	Elaboration Prompts (Example)
1	Writing OS-aware code.	What needs to be kept in mind when writing OS-aware code?	@Researcher, what is the advantage of writing OS-aware code like you did here?	@Driver, How would you improve the implemented approach? The Researcher can chime in too.
2	Need for pre-processing	Whitespace characters are removed while building the index. Why is this necessary?	Ok, @Navigator, can you say why these preprocessing steps are necessary before building the inverted index?	@Researcher, do you agree with this approach? @Project Manager, what do you think?
3	“Map” for word level count.	Which of these is true about combining word counts from different parts of a document?	@Driver, can you tell us how the word level counting is going to help us parallelize?	@Project Manager, do you think there is better implementation of this function? @Driver can chime in too about why.
4	“Reduce” for collating counts within document.	Which Spark code snippet produces an RDD of tuple (word, word frequency)?	@Researcher, can you think of another way to do the reduce?	@Navigator, what trade-offs do you think you made in this function implementation? @Driver, can you provide some insights too?
5	Collating across documents for inverted index.	What needs to be done to build a case-sensitive index?	@Driver, what did the groupBy operation achieve here?	Ok @Researcher, is there a better way to implement this? @Navigator, can you provide insight too?

Table 5.4: Learning objectives, examples of corresponding pre- and post test questions, information and elaboration prompts

Within each task, we specified four interdependent roles to better clarify the roles of the Mob members especially. As before, the Driver is the only participant who writes the code, based on high-level instructions received from Navigator, who makes decisions on the next course of action based on discussion with the rest of the team members. The Mob roles are defined further to support groups of size 4. The Researcher assists

the group in ideation and implementation by consulting resources such as a provided project primer and other external Web-accessible support material. The Project Manager is responsible for making sure that the rest of the team members are complying with and performing their roles adequately. After each programming task, the Project Manager reports on the level of compliance of the team and their progress of the project overall. As we saw in Chapter 2, prior research indicates that expert programmers need most help with the strategic aspects of language-independent planning [154]. These roles, much more refined than the typical ones in Mob Programming that we saw in earlier studies, serve distinct functions. While the Navigator engages in language-independent planning with support from the Researcher and Project Manager, they also engage in language-specific implementation planning with the Driver. Finally, the Project Manager explicitly helps smooth coordination issues for the group, thus reducing the likelihood of behaviors that might cause the breakdown of collaboration from manifesting.

Due to the differences in responsibilities associated with each of the roles, it is possible that there are varying degrees of learning associated with the roles. However, the roles rotate over the course of the task, making sure that any learning deficit is spread across the team and not concentrated on an individual. The rotation of roles also gives students the opportunity to contribute and observe different perspectives and approaches to solving problems in the same session. Finally, the interdependence baked into the roles requires the externalization of thinking in the course of discussing implementation alternatives, which provides opportunities for knowledge gaps to be revealed and addressed, a mechanism for individual learning from the collaborative learning environment [58, 178, 179]. A summary of the OPE roles is shown in Figure 5.3.

All students, regardless of the experimental condition, first participated in a training session two weeks before the experimental manipulation to familiarize themselves with the project environment. Students were placed into random groups of four based on their time availability and participated in sessions lasting 80 minutes, with roles switching every 8 minutes. The role-switching was kept relatively frequent in order to help students become accustomed to switching and performing the responsibilities associated with each role.

The experimental manipulation took place two weeks later, when the students had acquired the prerequisites for the project. As in the training session, students were randomly grouped into teams of 4 based on their time availability while making sure they were not placed into teams with students with whom they had done the training. The project lasted 80 minutes in total, with roles switching as each new task was successfully completed and the collaborative reflection associated with that task ended. This satisfies the need for milestone-based switching implied by the findings from prior studies in this series. Because the roles remained the same within each task (but varied systematically across tasks), we are able to measure the learning that occurred for each role during the task. Students in all of the conditions worked on 5 tasks, each contributing to an overarching project — an inverted index over a collection of Shakespearean texts in Scala. By default, they were given a maximum of 14 minutes to complete each task and discuss it before being asked to rotate roles. At the 14-minute mark, if they were

# Mob Programming Roles



**Driver** – Translates high-level group decision into code.



**Navigator** – Decides on one path forward by considering suggestions from the mob and conveys decision to driver.



**Researcher** – Refers to resources as necessary



**Project Manager** – Ensures on-task behavior of group members in the other roles.

Figure 5.3: A summary of the four roles that students are assigned to in the Online Programming Exercise (OPE)

still not finished, they were provided with a bottom-out hint to hasten the completion of the task.

## Experimental Manipulation

Based on the theoretical foundations of task design, the combination of task structuring and role scaffolding should bring the learning objective into focus for discussion and offer the opportunity for learning. However, there are many reasons why reflective discussion may not occur in a way that is conducive to learning.

First, students may choose not to focus and reflect on ideas that are meant to learn from the task. To address this, we design **information prompts**, which are reflection prompts that explicitly bring discussion topics into focus. They encourage the prompted student to examine and make explicit their mental model about the relevance of the task to the overall learning objective. In doing so, their articulation serves as an opportunity for others to examine their own mental models against the one presented and discuss to reconcile any emergent conflicts [21, 45, 59]. Because the prompts are designed for reflection on specific knowledge, they also serve as a form of direct instruction. Direct instruction is an efficient form of instruction and can lead to quicker knowledge convergence. However, it might not foster subject matter expertise that generalizes beyond the specific knowledge provided. Without meaningful practices for expanding on what is given explicitly, subject matter knowledge remains captured in internal scripts specific to tasks that could remain “inert” outside of those tasks [128].

Thus, a second concern is that students may discuss the target knowledge too briefly to elicit the deep reflection required for learning. To address this, we design **elaboration prompts** designed to draw out evaluation, elaboration, and exploration of alternative perspectives [129]. The elaboration prompts, in contrast to the information prompts, do not require a task-specific design but might result in additional time for reflection, which may adversely affect task efficiency.

Since project-based learning primarily focuses on task completion and the group product, an important locus of our investigation is whether learning is taking place.

We first investigate whether the redesigned OMP scaffold alone can produce a learning effect. This would mean that students are able to adequately manifest these learning behaviors on their own while being assisted by the agent only for role scaffolding.

- Hypothesis 1 — Students learn during programming — Learning occurs during the collaborative programming task, observed as gains from the pre- to post-test.

We will evaluate this hypothesis overall and specifically in conjunction with each of five separate learning objectives (LOs). Subsequently, we see if conversational agent-based prompts can offer benefits beyond the role scaffolding.

- Hypothesis 2 — Elaboration prompts improve learning — Elaboration prompts will serve to intensify the effect of task structuring and role scaffolding on discussion processes, thereby producing significantly more learning on the task. This will be observed in better pre-to-post-test gains (immediate) and better performance on a subsequent programming task (delayed).

- Hypothesis 3 — Information prompts improve learning — Information prompts will significantly improve learning either because they result in reflection or because they serve as a form of direct instruction.

We tested our hypotheses with a 2x2 factorial design in which the first factor was the presence or absence of information prompts, and the second factor was the presence or absence of elaboration prompts. In the conditions where the prompts were provided, students received them during the collaborative reflection upon completion of the programming task. Teams were randomly placed in the four conditions — 7 groups in the control condition where no prompts were presented, 6 groups presented with elaboration prompts, 5 with information prompts only, and 9 groups where both prompts were presented. The control condition checked to see if student-led support assisted by the task structuring and role assignment alone can bring the discussion into focus enough to impact learning. The two subsequent experimental conditions were tested to see if the elaboration prompts or the information prompts themselves would further intensify this learning effect. The final condition tested both prompts in tandem. Testing the effects of elaboration prompts with and without information prompts enables us to test whether the task structure and role scaffolding were sufficient to set the stage for collaborative reflection. An interaction effect where elaboration prompts only have an effect in conjunction with the information prompts would suggest a failure in the task structure and role scaffolding.

#### 5.4.2 Data Collection

In addition to the pre- and post-tests, performance on the subsequent individual project and productivity metrics for that project, such as the number of submission attempts (students could resubmit assignments repeatedly for autograding), serve as a delayed post-test measure. Finally, interaction logs including code contributions and chat logs were used to perform post-hoc process analyses to make sure that the levels of compliance to the role scaffolding were the same across conditions and thus could be eliminated as a potential confound.

#### 5.4.3 Analysis and Results

##### Hypothesis 1 — Students learn during programming

For each of the five LOs, students answered two questions worth 1 point each on the pre-test, and two questions, each worth 1 point, on the post-test. The average pre-test and post-test scores across the population for LOs 1, 2, and 5 were identical, which indicates no learning on average over the whole population. However, for LOs 3 and 4 there was a significant pre- to post-test gain as measured with a 2-tailed paired t-test,  $t = 2.43$ ,  $p < 0.05$  indicating that these two tasks lent themselves to learning during programming much more than the other tasks. For LO 2, average pre-test score was 1.7, post-test score 1.8, and standard deviation 0.6. For LO 3, average pre-test score was 1.5, post-test score 1.6, and standard deviation 0.6.

Thus, Hypothesis 1 is partially supported by significant gains before and after the test in LOs 3 and 4. Because of the learning gains achieved, we are able to test the other hypotheses regarding the intensification of learning specifically in connection with the two LOs where we observed significant pre- to post-test gains during the team programming project.

### **Hypothesis 2 — Elaboration prompts improve learning**

The model setup to test Hypothesis 2 and 3 was the same since the study was conducted as a factorial design. In particular, we used a repeated measures ANCOVA model, with LO and role as random variables, pre-test score (per LO) as a covariate, elaboration prompts and information prompts and the interaction between the two as independent variables and post-test score (per LO) as the dependent variable. As an aside, there was no statistically significant difference in learning between roles.

In terms of pre- to post-test gains, there was no significant main effect of the elaboration prompt factor;  $F(1, 440) = 0.21, p = n.s.$  However, there was a significant interaction effect between the two experimental factors  $F(1, 440) = 11.6, p < 0.0001.$  In a post hoc analysis, we determined that both conditions with only one type of prompt were associated with significantly more learning than the control condition, and the condition with both types of prompts was not significantly different from the control. The effect size of the addition of elaboration prompts over no prompts was 0.32 SD, which is a medium effect size.

To test the impact on a subsequent individual programming task, we built an ANOVA model, with elaboration prompts and information prompts and the interaction between the two as independent variables, to measure the impact of the experimental manipulation separately on three outcome measures related to task performance — time on subsequent programming task, number of submitted attempts on that task, and score. Here, there was a trend for the elaboration condition to improve performance in terms of time-on-task, number of submission attempts, and score, though none of these were statistically significant.

Thus, Hypothesis 2 is supported for immediate learning from the task, but not in the delayed test, although positive trends were observed.

### **Hypothesis 3 — Information prompts improve learning**

As mentioned above, we observed an interaction effect in which each prompt type alone led to significant gains over having neither type of prompt, but not when they were used together. The effect size of the addition of information prompts over no prompts was 0.42 SD, which is a medium effect size. With respect to the subsequent individual programming task, there were no significant or marginal main effects of information prompts or interaction effects; however, the trend was consistently that information prompts were associated with lower time-on-task, lower number of submissions, and higher scores. Thus, Hypothesis 3 is supported for immediate learning from the task but unsupported on the delayed test, although positive trends were observed.

#### 5.4.4 Discussion

We observe significant pre- to post-test gains for LOs 3 and 4. Tasks where the focus is on programming language syntax and therefore on procedural knowledge seem to provide less scope for multiple perspectives to manifest in the discussion. Tasks 3 and 4 were more focused on the correct implementation once students were already made aware of the syntax through earlier tasks, and resulted in richer discussions providing a possible explanation for learning in these tasks. Using either the elaboration prompt or the information prompt alone significantly improved learning over the control condition. The use of both together was counterproductive. In order to better understand the mechanism and the results, we performed a process analysis over text responses during the collaborative reflection.

We observed that when the agent first presents the prompts, students are often still engaged in a discussion about the task. It is only after this discussion concludes that they turn their attention to the prompts. In the case where an information prompt alone was presented, the prompted student engaged in the reflection in 60% of the cases for LO 3 and 50% of the cases for LO 4 while the rest of the group observed. Discussion ensued if the group disagreed with the student's response. In other words, one student's reflection served as a trigger for other students to examine their own mental model and reconcile them through discussion if conflicts emerged, thus leading to learning. This is consistent with the theoretical mechanism for collaborative learning described in Chapter 2.

In the case where the elaboration prompt alone was presented, the prompted students engaged in 40% of the cases for LO 3 and 100% of the cases for LO 4. The groups were observed to engage in the discussion, explicitly elaborating on each other's ideas when directed by the agent. When both prompts are presented, however, the agent's prompts appeared to treat the ongoing conversation as a response and the subsequent prompt as a response to that, which would be incoherent and, therefore, break the illusion that the students are obliged to respond. Once this happened, students ceased to engage with the prompts for the duration of the task and chose to focus on the implementation instead. In other words, because the prompts do not result in engagement or discussion, this condition is equivalent in effect to the condition where no prompts were presented.

Neither type of prompt had a significant positive effect on delayed post-test performance (the individual project after the manipulation), though the results in connection with both types of prompts showed positive trends consistent with prior work [115]. It is notable that both types of prompts, when used on their own, had a medium effect size but the elaboration prompts may have practical advantages from the standpoint of developing scaffolds generally as they do not require task-specific tailoring.

## 5.5 Chapter Discussion and Lessons Learned

A summary of the design-based research iterations over the four experimental studies presented in this chapter is shown in Table 5.5.

	Context	Learning Measure	Group Size	Roles	Role-Rotation Strategy
1	6-week Working Professionals Course	N/A	3-6	Driver, Navigator, Mob	Time-Based
2	Semester-long Higher-Ed Course	Average of Assessments Prior to, and After Project	3	Driver, Navigator, Mob	Time-Based
3	Semester-long Higher-Ed Course	Average of Assessments Prior to, and After Project	4	Driver, Navigator, Mob	Time-Based
4	Semester-long Higher Ed Course	Pre-, Post-, and Delayed Post-Tests	4	Driver, Navigator, Researcher, & Project Manager	Milestone-Based

Table 5.5: Summary of experimental studies in the design-based research iterations.

Through a series of four experimental studies, we iteratively refined the design of a team programming project to better support student learning. Several key lessons emerged that informed the final design —

### 5.5.1 Alignment with KLI Framework

As discussed in Chapter 3, we use the KLI framework to define granular learning objectives for the programming project. The evolution of the experimental design reflects a growing understanding of how to align knowledge components, learning processes, and instructional principles, as defined in the Knowledge, Learning, and Instruction Framework [88] over the course of the four design-based research iterations —

- Initially, the knowledge to be learned was not clearly defined, making it difficult to measure learning gains. By the final study, we identified specific learning objectives for each task, allowing for more precise measurement of learning outcomes.

- Early studies focused primarily on collaborative behaviors rather than cognitive processes. Later experiments incorporated prompts designed to elicit specific learning processes, such as elaboration and reflection.
- The initial design relied heavily on the industry-inspired Mob Programming structure. Subsequent iterations introduced additional scaffolding, including refined role definitions, conversational agent prompts and task structuring aligned with learning objectives.

### 5.5.2 Key Design Improvements

1. Uptake of the Scaffold — The first study demonstrated that students could follow prescribed Mob Programming roles, but this alone was insufficient to guarantee learning.
2. Comparison to Self-Organization — The second study revealed that the Mob Programming scaffold produced more distinct collaborative behaviors compared to self-organized groups.
3. Need for Task Design — The third study finalized the group size, tested an implicit form of scaffolding in tandem with the explicit role scaffolding, and compared performance to individuals working through the project on their own. While it produced correlational evidence of an increased conceptual focus and an even labor distribution, the lack of fine-grained measurement preventing directly measuring learning from the task.
4. Fine-grained Measurement — The final study implemented pre- and post-tests aligned with specific learning objectives, allowing for more precise measurement of learning gains.
5. Targeted Prompts — The introduction of information and elaboration prompts in the final study led to significant improvements in immediate learning outcomes for some tasks.
6. Role of Reflection — The final study revealed that collaborative reflection after task completion was more conducive to learning than reflection during problem-solving.
7. Task Difficulty Balance — Across the studies, we found that task difficulty needed to be carefully calibrated to provide both challenge for multiple perspectives to manifest and achievability within time constraints.
8. Group Size Considerations — The first study explored various group sizes, ultimately settling on groups of four as optimal for balancing participation and manageability. This was supported by refined role definitions for Mob members.

### 5.5.3 Final Design Alignment with KLI

The final experimental design showed an improved alignment with the KLI framework.

- Knowledge Components — Clearly defined learning objectives for each task.
- Learning Processes — Incorporating elaboration and reflection prompts eliciting specific cognitive processes.
- Instructional Principles — Structured collaboration through Mob Programming roles, combined with targeted prompts and task design aligned with learning objectives.

This iterative design process demonstrates the value of applying theoretical frameworks such as KLI to guide the development of effective learning environments. By progressively refining the alignment between knowledge components, learning processes, and instructional principles, we were able to create a more effective learning environment for advanced computer science courses.

Based on the results of the final study, we identified that tasks focused on conceptual aspects of implementation, rather than syntax, were more conducive to learning. The literature on example-based learning suggests that reflection based on worked examples is far better for this kind of learning than problem-solving. Consequently, the subsequent chapter will focus on introducing worked examples within the team programming project and compare its efficacy to the use of problem-solving practice, that is, programming alone.

#### 5.5.4 Lessons for Practitioners

These experiments provide a series of lessons for practitioners so that they can avoid costly implementation iterations and design effective projects from the get go.

- Align with Learning Objectives — Carefully define the specific learning objectives for each task and ensure that they are measurable. This allows for a more precise evaluation of learning gains.
- Design for Measurement — Instrument the project to collect fine-grained data on both the process (e.g., chat logs, code contributions) and outcomes (e.g., pre/post-tests aligned with learning objectives).
- Scaffold Collaboration — Implement explicit scaffolding for collaboration, such as defined roles with clear responsibilities. This promotes more even participation and focused discussions.
- Optimize Group Size — Aim for groups of 4-5 students. This size balances participation opportunities with manageable coordination.
- Calibrate Task Difficulty — Design tasks that are challenging enough to require collaboration but achievable within time constraints. “Collaboration-worthy” problems are key.
- Incorporate Reflection — Include structured reflection opportunities after task completion to enhance learning.
- Use Targeted Prompts — Implement information and elaboration prompts to guide discussions toward learning objectives. However, use them judiciously to

avoid cognitive overload.

- Balance Learning and Productivity — Be mindful of students' tendency to prioritize task completion over learning. Design activities and assessments that reward both outcomes.
- Provide Role Training — Offer initial training sessions to familiarize students with collaborative roles and tools before the main project.
- Iterative Refinement — Be prepared to refine the design based on data and feedback. Although these lessons can help avoid major pitfalls, some iteration may still be necessary to optimize for specific contexts.

By incorporating these lessons from the start, practitioners can design more effective collaborative programming activities that promote learning while avoiding many of the challenges encountered in the iterative process described in this research.

### **5.5.5 Lessons for Education Researchers Conducting Data-Driven Experiments**

The series of studies described in this chapter offers valuable insights for education researchers conducting data-driven experiments as well —

- Iterative Design Process — Embrace a design-based research approach with multiple iterations. Each study should build on the findings of the previous one, allowing for continuous refinement of the experimental design and instructional interventions.
- Alignment with Theoretical Frameworks — Ground experimental design in established theoretical frameworks, such as the Knowledge, Learning, and Instruction (KLI) framework. This alignment helps in clearly defining knowledge components, learning processes, and instructional principles.
- Granular Learning Objectives — Define specific, measurable learning objectives for each task or component of the intervention. This granularity allows for more precise measurement of learning outcomes and identification of where learning is occurring.
- Multiple Measurement Points — Implement pre-, post-, and delayed post-tests to capture both immediate and long-term learning gains. This approach provides a more comprehensive view of the effectiveness of the intervention.
- Process Analysis — Collect and analyze process data (e.g., chat logs, code contributions) in addition to outcome measures. This data can provide insights into the mechanisms behind observed learning effects and help explain unexpected results.
- Controlled Comparisons — Design experiments that allow for controlled comparisons between different interventions or conditions. The 2x2 factorial design in the final study is a good example of this approach.
- Consideration for Context — Be mindful of the educational context in which the

experiment is conducted. Constraints such as location, availability, and duration, that were considered in these studies are examples of context-specific factors.

- Unexpected Interactions — Be prepared for unexpected interactions between different interventions. The final study revealed that combining information and elaboration prompts was less effective than using either type alone, highlighting the complexity of designing multi-faceted interventions.
- Mixed Methods Approach — Combine quantitative analyses with qualitative insights from student feedback and behavior observations. This mixed-methods approach can provide a richer understanding of the intervention's effects and student experiences.
- Practical Implications — Consider the practical implications of research findings for instructional design and classroom implementation. The studies progressively moved towards interventions that could be more easily integrated into existing course structures.

By following these lessons, researchers can design more robust, theoretically grounded, and practically relevant experiments in data-driven education research.

# Chapter 6

## Problem-Solving versus Worked Examples

While redesigning the team programming project from the ground up produced a version that resulting in conceptual learning gains, we also see that guidance provided by conversational agent-based prompts can magnify those gains. Testing even more guidance, such as with worked examples, would be the next logical step. However, the concern from instructors in this context is that worked examples alone, while better for conceptual learning, may leave students less able to implement those concepts in computer programs. Consequently, in this chapter, we establish a comparison of problem-solving practice with worked examples based not only on their impact on conceptual learning, but also on individual project performance.

Informed by the literature on example-based learning [127], we present a research agenda in this chapter to compare the use of worked examples to problem-solving practice in a context where the domain-specific concepts that students are learning are new, but substantial foundational knowledge about programming has been acquired in their prior learning experiences. Even in this advanced stage at the macro-curricular level, we hypothesize that students will benefit from collaborative reflection on worked examples over problem-solving since they are learning new concepts and are in the early stages of acquisitions of the skills related to them. Our findings are consistent with the literature on example-based learning, but challenge widely-held views about what "new" and "early" mean for the computer science curriculum. As elaborated in Chapter 2, students in this context have domain-relevant conceptual knowledge, as well as declarative, procedural, and conceptual programming knowledge in at least one programming language. They may lack procedural knowledge depending on the programming language used for the project, and they need help with the strategic aspects of language-independent planning for implementation in addition to domain-specific conceptual learning.

## 6.1 Study 1 — Problem-Solving versus Practice versus Worked Examples

### 6.1.1 Course Context and Experimental Design

For the team programming project, the task structuring macroscript first divides the project into tasks, each aligned with a learning objective. Each task is further divided into a problem-solving phase and a collaborative reflection phase. Since students cannot simultaneously attempt multiple tasks, the task-structuring macroscript discourages their divide-and-conquer tendencies and encourages them to work collaboratively on the task at hand. During the problem-solving phase of the task, the collaboration is additionally structured using the role scaffolding macroscript based on the Mob Programming paradigm [199, 204]. Roles that students are assigned to, designed for an online instructional context based on this paradigm [136, 142], require the explicit (and public) articulation of ideas laying the foundation for transactive exchange to occur. The Driver, Navigator, Researcher, and Project Manager roles were used, based on the final design presented in Chapter 5. Together, the task structuring and role scaffolding macroscripts serve to foster a learning orientation during the problem-solving phase of the task while explicitly discouraging a divide-and-conquer approach. During the collaborative reflection phase, then, an intelligent conversational agent provides discourse-level microscript prompts that draw the students' joint attention to the aspects of the problem that are relevant to the learning objective. These prompts can support specific discourse activities such as building arguments, supporting learners in justifying their claims, or explicitly prompting another learner to build on an existing argument toward knowledge construction [166]. The programming practice and worked examples provided during the problem-solving phase serve to familiarize students with the problem context, while the conversational agent prompts during the collaborative reflection phase served to draw their attention to relevant problem states. Since the scaffolds play complementary roles, we might see a synergistic effect of the two in this context [146].

This study was conducted in the Fall 2020 offering of the Cloud Computing course<sup>1</sup>. Our experiment is situated within the third sub-unit of the third project unit of the course that focuses on "multi-threaded programming and consistency". In preparation for the individual project in this unit, students, in groups of 4, work on the OPE. Pre- to post-test score improvements serve as a measure of conceptual learning from each task. The pre- and post-test scores for each LO is the average of the scores of the two questions targeting that LO, and thus ranges between 0 and 1. Performance on the individual project corresponding to this sub-unit serves as a procedural and conceptual problem-solving transfer task (delayed post-test). This project is graded by the instructor on a rubric with 12 quality scores, each of which ranges between 0 and 1.

Table 6.1 shows the learning objectives and examples of pre- and post-test questions corresponding to each task, while Figure 6.1 shows the position of pre-, post-, and delayed post-tests within the course. A summary of the course structure and the location

<sup>1</sup><http://www.cs.cmu.edu/~msakr/15619-f20/>

of the study within it is shown in Figure 6.1. A total of 74 students completed the exercise and the subsequent project.

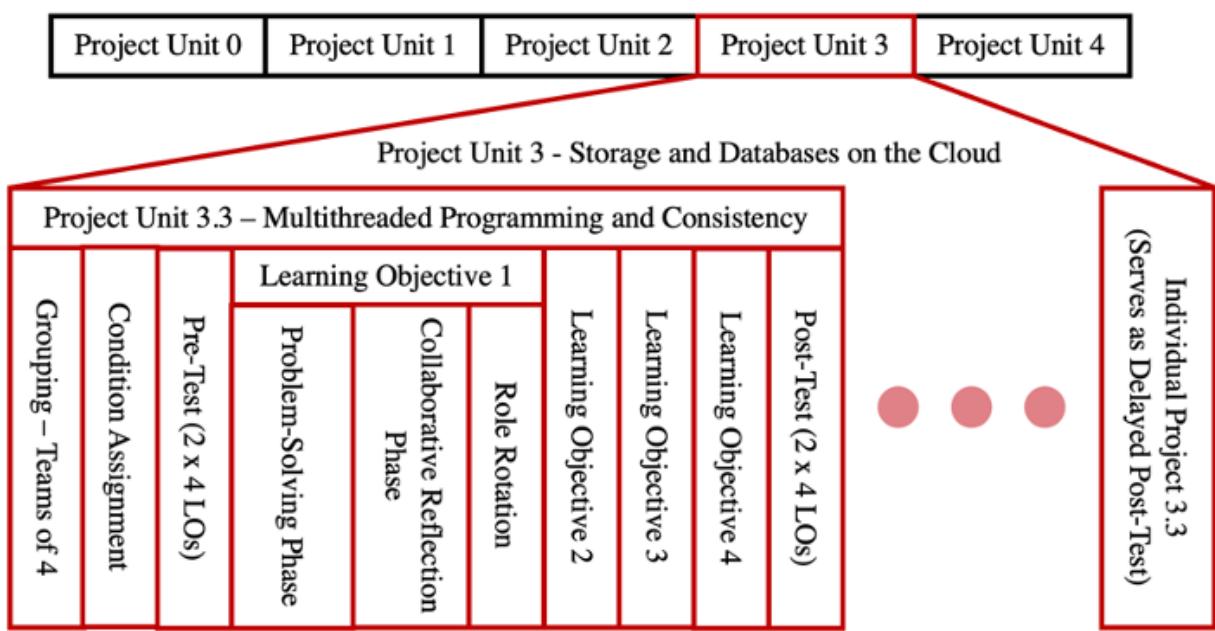


Figure 6.1: Course structure, pre-test, post-test, and delayed post-test alignment.

Since the course had been offered online for over 15 semesters, the course content and structure did not need to be changed in response to the move to online learning in the wake of the COVID-19 pandemic. The main effect was seen in the enrollment numbers which were about half the usual. Some students participated from off-campus locations in other time zones. Several options of time slots, amenable to students from different time zones, was provided to facilitate participation in the synchronous team programming projects.

## Experimental Manipulation

Two weeks before the experimental manipulation, students participated in a training session in randomly formed teams of 4 based on their time availability. They were first provided with videos and reading materials explaining the OPE and motivating its use for team programming projects. For the practice project, they worked on a data processing task using the pandas library in Python. The project was relatively simple but still formed a meaningful component of the course. It was kept simple to provide students the opportunity to familiarize themselves with the OPE — the roles, the role rotation, and the interface of the AWS Cloud9 Collaborative IDE used for the project. Keeping it meaningful meant that students took the project seriously. Prior iterations observed that the lack of training led to deleterious perceptions of the actual team programming projects, and that lack of training may have been due to the task being so simple that students simply skipped to the end without gaining any practice. As before, each project

#	Learning Objective	Example Pre/Post-Test Question - Multiple-Choice	Example Collaborative Reflection Prompts
1	Building blocks of a multi-threaded application	Which of the following statements about multithreading in Java is INCORRECT?	Was your approach similar to the reference solution? What Thread class functions did you use? Take turns explaining the logic for what you did here.
2	Diagnosing and fixing deadlocks	The usage of notify() will never result in a deadlock in which of the following multithreaded application scenarios?	In an ideal scenario, can you think of a built-in Java thread-safe class that could replace the priority queue? Take turns explaining how this might work.
3	Diagnosing and preventing a race condition	Examining the following code snippets identify the one that will NEVER lead to a race condition.	Comparing your approach to the reference solution, how did you avoid the race condition here? Take turns explaining the logic.
4	Ensuring strong consistency in data stores	How would you acquire a lock on a critical resource shared by multiple threads to ensure consistent runtime behavior?	Can you put what you are learning in all these tasks together to think about ensuring strong consistency? Take turns explaining.

Table 6.1: Learning Objectives, Corresponding Pre/Post Test Questions (Examples), Information and Transactivity Prompts

session lasted for a total of 80 minutes.

Two weeks after the OPE training session, when students had acquired the prerequisites necessary to complete the team programming project for the unit, they were grouped randomly into teams of 4 based on their time availability while making sure they were not placed into teams with students they had done the training with. Each project session, once again, lasted a total of 80 minutes with roles switching as each new task was completed, and the collaborative reflection phase associated with that task ended. The 74 students were assigned to 19 teams of which 17 were 4-member teams and 2 were 3-member teams. In the 3 member teams, the student assigned to the project manager role also acted as the researcher. 9 teams were assigned to the maximize learning from problem-solving (MLPS) condition, where, for each task, teams complete the problem-solving and then enter into a reflection phase for the remaining time, and 10 teams were assigned to the maximize learning from reflection (MLR) condition, where problem-solving is curtailed after a pre-specified amount of time, and they enter the reflection regardless of whether they completed the problem-solving or not.

- Hypothesis 1 — The Online Programming Exercises (OPEs) result in pre- to post-test learning gains — These exercises discourage students' divide-and-conquer tendencies and promote transactive exchange for learning.
- Hypothesis 2 — The maximize learning from reflection (MLR) condition will result in better pre- to post-test learning gains — Since the MLR condition provides a stronger focus on the conceptual learning, pre- to post-test learning gains which measure conceptual learning from the task will be significantly higher in this condition.
- Hypothesis 3 — The maximize learning from problem-solving (MLPS) condition will result in better performance on the individual project — Since the MLPS condition provides more problem-solving practice, students in this condition will perform better on the individual project which serves as the delayed post-test of problem-solving transfer.

### 6.1.2 Analysis and Results

#### **Hypothesis 1 — The Online Programming Exercises (OPEs) result in pre- to post-test learning gains**

To evaluate the general value of the team programming project for conceptual learning, regardless of condition, we compare pre- and post-test scores per learning objective, role, and condition, where pre- and post-test scores vary between 0 and 1 per learning objective. For this analysis, we build an ANOVA model with test score as the dependent variable, and time-point (pre- vs post-test), condition (MLPS vs MLR), role (Driver, Navigator, Researcher, Project Manager), and learning objective (listed in Table 6.1) as independent variables. We also included pairwise interaction terms between time-point and each of the other three independent variables. There was a significant effect of time-point  $F(1,410) = 3.77$ ,  $p < 0.0001$ , effect size 0.38 SD, with an average pre-test score of 0.55 (.37 SD) and average post-test score of 0.69 (.37 SD). None of the pairwise interactions were significant. Thus, we confirmed that students in both conditions learned based on the significant difference between pre- and post-test scores across the two conditions regardless of role or learning objective. Thus, the Hypothesis 1 is confirmed.

#### **Hypothesis 2 — The MLR condition will result in better pre- to post-test learning gains.**

In order to test the effect of condition on the magnitude of learning we compare post-test scores between conditions controlling for pre-test scores. In particular, we computed an ANCOVA model with post-test score as dependent variable, pre-test score as the covariate, and condition (MLPS vs MLR), and learning objective as independent variables. We found a significant effect of condition such that students in the MLR condition learned more  $F(1,254) = 6.0$ ,  $p < 0.05$ , effect size 0.24 SD. For the MLR condition, the average pre-test score was 0.53 (0.36 SD) and post-test score was 0.72 (0.34 SD), and for the MLPS condition, average pre-test score was 0.57 (0.38 SD) and post-test score

was 0.65 (0.37 SD). We find significantly higher pre- to post-test gains in the MLR condition in comparison with the MLPS condition ( $p < 0.05$ ), which suggests that students with domain expertise benefit more from the worked example-based reflection than the problem-solving for acquisition of new conceptual knowledge. Thus, the Hypothesis 2 is confirmed.

### **Hypothesis 3 — The MLPS condition will result in better performance on the delayed post-test**

In order to test the effect of condition on performance on the individual project, we considered each of the 12 quality ratings assigned by the instructor within a single model in order to control for multiple comparisons. In particular, we computed a single ANCOVA model with numeric quality rating as the dependent variable, total pre-test score across learning objectives as a covariate, and the condition and the name of the quality rating as independent variables. We also include the pairwise interaction term between the two independent variables. There was a significant effect of condition such that students in the MLR condition scored higher than students in the MLPS condition,  $F(1,707) = 4.36$ ,  $p < 0.05$ , effect size 0.15 SD, which is a weak effect. The average score was 4.7 (2.7 SD) for the MLPS condition and 5.0 (2.5 SD) for the MLR condition. There was no significant interaction between condition and quality rating name. Thus, although the effect is weak, students in the MLR condition performed better than students in the MLPS condition across the 12 quality ratings. The third hypothesis is rejected, and in fact, the opposite is weakly supported. Contrary to expectation, not only is worked example-based reflection superior for conceptual understanding, it is better preparing students for performance on the individual project as well.

#### **6.1.3 Discussion**

This first study contrasted two conditions —} maximize learning from problem-solving (MLPS) and maximize learning from reflection (MLR).

To start, we find that the team project exercises lead to significant pre- to post-test learning in both conditions. This reiterates the success of the project redesign undertaken as a part of the design-based research iterations presented in Chapter 5. Further, worked example study and problem-solving practice are both potentially valuable for conceptual learning from the project. While we did not compare against either of the scaffolds provided on their own, the lack of a detrimental effect in the two conditions we do test means that the scaffolds were not so redundant as to draw student attention away from the relevant problem states. The problem-solving practice and worked examples served to acquaint students with the problem context while the collaborative reflection scaffolded by the conversational agent drew their joint attention towards the relevant problem states indicating that the scaffolds played potentially complementary roles.

When comparing across conditions, we see that the condition where students spent less time on problem-solving and more time on worked example-based reflection re-

sulted in significantly more pre- to post-test learning. Based on cognitive load theory, we could surmise that it is indeed the case that extensive problem-solving consists of production steps that are superfluous to the learning here. While problem-solving practice was not detrimental to student learning, we can more efficiently use their time and impact their learning more significantly if we use worked examples as well, with an emphasis on time spent on reflecting rather than problem-solving. In other words, the worked examples were more successful than problem-solving practice in drawing student attention to problem states relevant to the learning objective.

One concern among educators has been that while students' conceptual learning can be positively impacted by the use of worked examples, problem solving transfer may be negatively impacted from the lack of practice during the team programming project. We started with this hypothesis and we would have not been surprised to have found that. However, what we found was support for the opposite. Students who reflected longer also performed better on the individual project which served as a delayed post-test measure of problem-solving transfer, though the effect size was small. We can conclude that students, at this point in the course, had already acquired the procedural aspects of programming knowledge enough to not need the practice. In other words, given a schema, they were able to translate that into a solution to the problem. The positive impact of the worked example condition on the conceptual process of schema acquisition and induction then led to a positive impact on student performance on the subsequent project also.

In looking at the chat logs to discern the potential reasons for these findings, we see that the students in the MLPS condition were indeed involved in more discussion related to the syntax and procedural aspects of implementing a working program. Even after they arrived at a workable solution, they spent a non-trivial amount of time to get the implementation right instead of discussing and better understanding the conceptual aspect of the solution. The part played by the student in the researcher role is particularly illustrative of this difference. In the MLPS condition, the researcher was involved in "research" about the correct syntax and possible in-built functions to use in their endeavor to help the group implement a correct, working solution. In the MLR condition, they were tasked with helping the group recall relevant concepts from the primer in order to respond to prompts from the agent. This deep-dive shows that the findings are consistent with the literature on example-based learning. For the students in this advanced curricular context, the procedural aspects of programming are indeed superfluous, and their time is better spent on learning the conceptual aspects of programming.

In the following experiment, we expand on this research agenda by exploring the trade-off between worked examples and problem-solving more directly, by varying the boundary between the two while maintaining total time on task constant. This allows us to determine the right balance of the two scaffolds that maximizes conceptual learning as well as problem-solving transfer from. After that, we investigate the sequencing effect of using collaborative reflection as a means to prepare students for problem-solving and compare that against collaborative reflection based on problem-solving practice. Taken together, these experiments provide a comprehensive comparison of the two scaf-

folds.

## 6.2 Study 2 — Time-Distribution between Problem-Solving and Worked Examples

### 6.2.1 Course Context and Experimental Design

This study was conducted in the same course, but in the first sub-unit of the fourth project unit of the course that focuses on “building inverted indices in Scala”. A summary of the course structure and the location of the study within it is shown in Figure 6.2. A total of 48 students participated in the team programming project and the subsequent individual project from across two campuses.

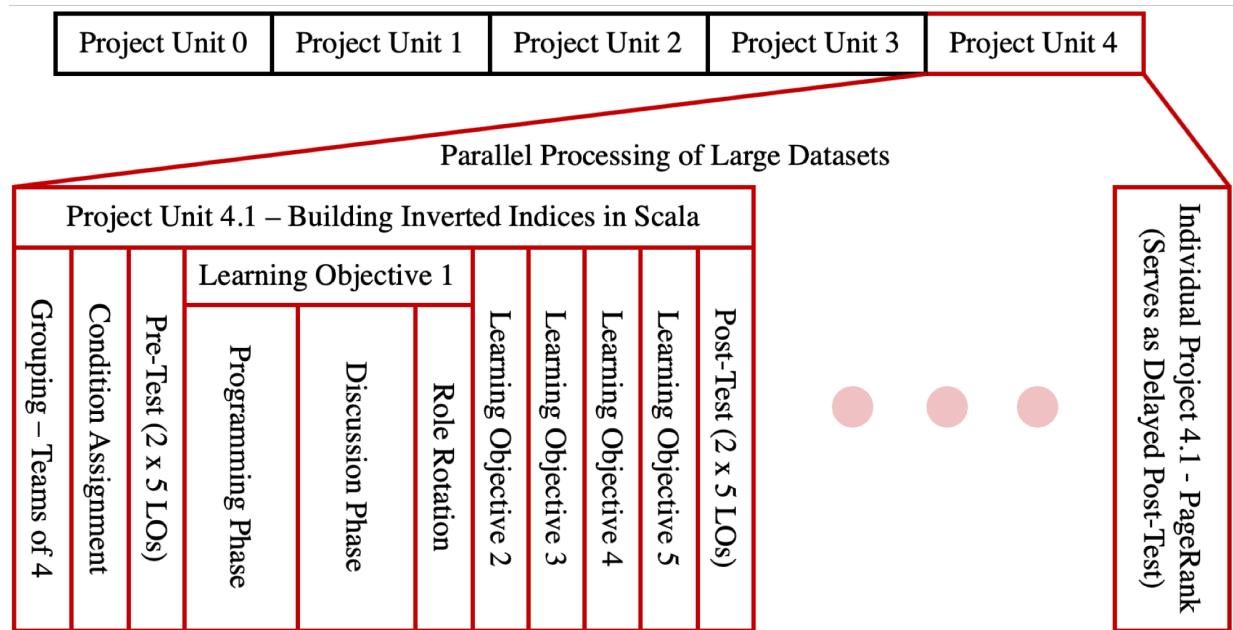


Figure 6.2: Course structure, pre-test, post-test, and delayed post-test alignment.

The unit introduces students not only to the conceptual aspects of inverted indices and the PageRank algorithm, but also to the procedural aspects of thinking in, and implementing concepts learned in a new programming language, Scala. Scala uses a programming paradigm called functional programming that most students are unfamiliar with prior to this unit. Consequently, this becomes the ideal context to test the differential impact of problem-solving practice in the form of collaborative programming, and worked example study in the form of collaborative reflection, on conceptual learning from the task, and performance on the subsequent problem-solving transfer task. The declarative aspects of Scala are taught in the primers, leaving the lack of prior procedural knowledge in Scala as the difference dimension when compared to the previous study.

The project design is the same as the previous experiment. The only difference is that the boundary between the problem-solving and collaborative reflection phases is able to be adjusted to study the effect of time distribution between the two treatments. Table 6.2 shows examples of these collaborative reflection prompts.

#	Learning Objective	Example Pre/Post-Test Question (Multiple-Choice)	Example Collaborative Reflection Prompt
1	Utilize the suitable map and reduce operations in Spark to transform and aggregate data.	Examine the following ways of computing the sum by key of the exampleRDD. Which of them is the most efficient?	Think of the solution you were attempting. Is this the same as what the bot presented in the chat? Take this time to explain the solution to each other.
2	Utilize suitable functional operations in Scala to transform data structures.	Which of the following is a right way of transforming srcList to destList?	There are a few different ways of doing element indexing such as dot indexing or the scala case() function. Which of these is shown in the example and which would be better aligned with best practice?
3	Identify the use case of a JOIN operation and utilize JOIN to combine and transform different RDDs in Spark.	Calling JOIN on two RDDs of type (K, V1) and (K, V2) results in which of the following types?	How is the combining of values implemented here? Can you unpack the working based on the example presented?
4	Identify the use case of the cache() method and utilize it to improve the performance of a Spark application.	What is caching in memory meant to achieve in Spark?	What factors do you need to keep in mind while deciding what to cache? See if you can discover the list of factors from the example.

Table 6.2: Learning objectives, and examples of corresponding pre- and post-test questions and collaborative reflection prompts

The individual project contains six tasks that map directly to the learning objectives of the team programming project. Table 6.2 shows the learning objectives and examples of pre- and post-test questions corresponding to each task, while Figure 6.2 shows the position of pre-, post-, and delayed post-tests within the course.

## Experimental Manipulation

For the experimental manipulation, students, in groups of 4, were assigned to 4 different conditions as follows —

- Mostly problem-solving — Students spend two-thirds of the time on collaborative problem-solving before being cut-off and presented with a worked example

solution for collaboration reflection.

- Equal problem-solving and worked example discussion — Students spend equal amounts of time on collaborative problem-solving and reflection based on the presented worked example.
- Mostly worked example discussion — Students spend one-third of the time on problem-solving and the rest of collaborative reflection.
- Only worked example discussion — Students spend all of their time reflecting based on the presented worked example.

The total project session lasted 80 minutes. 5 minutes is allocated in the beginning for setup and introductions. Another 5 minutes is allocated at the end of the session for students to submit their solution. 10 additional minutes was considered the buffer counting the time for context-switching from problem-solving to worked example discussion and rotating roles. The remaining 60 minutes was for the actual tasks during which the scaffolds were split according to the conditions that students were assigned to, as shown in Table 6.3. The 48 students were placed in groups of 4 with 3 groups of students — 12 students in each of the four conditions [147].

Task # \ Condition	Mostly problem-solving	Equal	Mostly worked example	Only worked example
1	12 + 6	9 + 9	6 + 12	0 + 18
2	12 + 6	9 + 9	6 + 12	0 + 18
3	8 + 4	6 + 6	4 + 8	0 + 12
4	4 + 2	3 + 3	2 + 4	0 + 6

Table 6.3: The boundary between collaborative programming practice and worked example discussion across tasks. Legend: a + b, where 'a' indicates time spent on collaborative programming, and 'b' represents the time spent on collaborative reflection.

An example student interaction facilitated by the conversational agent during the collaborative programming phase is shown in Figure 6.3. An example interaction during the collaborative reflection phase is shown in Figure 6.4.

The individual project that served as the problem-solving transfer task was due a week after the team programming project.

- Hypothesis 1 — The Online Programming Exercises (OPEs) result in pre- to post-test learning gains — Regardless of whether students participate in more problem-solving or worked example discussion, the project results in pre- to post-test learning gains overall.
- Hypothesis 2 — Worked example discussion contributes more to conceptual learning as measured by pre- to post-test learning gains than problem solving. Thus, we expect the most learning where students maximize time on worked example study, and a reduction in that learning as students spend more of their time on problem solving.

OPE\_Bot: "Ok, we are switching roles now. The new task is unlocked."

OPE\_Bot: "The new roles are -

S1 - Navigator

S2 - Driver

S3 - Researcher

S4 - Project Manager"

S3 (Researcher): "@S1 (Navigator), what could (sic) I help you find?"

S1 (Navigator): "Are we supposed to use only the helper functions?"

S3 (Researcher): "Yes, I think so! Looked at it (the task writeup."

...

S4 (Project Manager): "@S2 (Driver), want to try implementing this in that block with our variables?"

S2 (Driver): It has some build errors.

....

Figure 6.3: Example student interaction facilitated by the conversational agent during the collaborative programming phase.

- Hypothesis 3 — Problem-solving practice contributes the most to performance on the individual programming project measure of problem-solving transfer. Thus, we expect students with more problem-solving practice to perform better on the problem-solving transfer task.

## 6.2.2 Analysis and Results

### Hypothesis 1 — The Online Programming Exercises (OPEs) result in pre- to post-test learning gains across all conditions

We first tested to ensure that there was significant pre- to post-test learning across all conditions. For this analysis, we built an ANOVA model with Time Point (pre- vs post-), Learning Objective, and Condition as independent variables, as well as all two and three

way interactions, and test score as the dependent variable. There was a main effect of time point such that test scores were higher on average at post-test time  $F(1, 372) = 14.4$ ,  $p < 0.0001$ , effect size 0.34 SD. None of the two or three way interactions were significant. Therefore, students learned for all learning objectives in all conditions between the pre- and post-test. Thus, Hypothesis 1 is fully supported.

### **Hypothesis 2 — Worked example study contributes more to conceptual learning as measured by the pre- to post-test learning gains than problem solving**

In order to test the second hypothesis, we built an ANCOVA model with post-test score as the dependent variable, pre-test score and percentage of time spent on problem solving as covariates, and Learning Objective as an independent variable. In this analysis, there was a significant negative correlation between percentage of time spent on problem solving and post-test score in this analysis, indicating that as we manipulated amount of time spent on problem solving, the more time spent on problem solving, the less learning took place.  $F(1, 182) = 5.42$ ,  $p < 0.05$ . The partial correlation accounting for just the effect of percentage of time spent on problem solving is  $R=0.17$ . When we treat Condition instead as a nominal variable, there is no significant effect of condition. However, there is a trend in favor of less problem solving ( $p = 0.14$ ) and a student-t post-hoc analysis shows that students in the condition with no problem solving learned significantly more than students in the maximum problem solving condition (effect size 0.39 SD), with the other two conditions falling in between, and not being significantly different from the other two. Thus, Hypothesis 2 is supported, though the effect may be weak.

### **Hypothesis 3 — Problem-solving practice contributes the most to performance on the individual programming project measure of problem-solving transfer**

In order to test the third hypothesis, we constructed a repeated measures dependent variable that comprises the 6 separate scores assigned to evaluate the quality of the individual project. The Construct variable distinguished between the 6 different criterion scores. We built an ANCOVA model with pre-test score and percentage of time spent on problem solving as covariates and Construct as the independent variable. The criterion score for each of the Construct labels was the dependent variable. There was no effect of Construct or Condition. We also tested the model with percentage of time nested within Construct in case the condition variable had a differential effect across construct levels, but there was still no effect. Thus, Hypothesis 3 was not supported. What this means is that not only did students learn more conceptual knowledge from spending more time studying worked examples, they did not incur any deficit in terms of their ability to perform on the individual project, either overall, or on any single criterion score.

The verbose results from the analyses are shown in Table 6.4.

OPE\_Bot: "You have run out of time for this test case.  
Don't worry! Here is a reference solution - "

OPE\_Bot: .... (Reference Solution)

OPE\_Bot: "Can you compare the solution you were  
attempting to the example shown?"

S1: "Oh it uses join."

S2: "Yes..." (Still viewing the example)

S3: "and it uses pattern matching here"

...

S4: "I guess zip should work too?"

S1: "Yeah I think so."

S1: "here if we use indexing it would a bit confusing I  
guess" (sic)

...

OPE\_Bot: Based on your solution and the example, which  
implementation is better aligned with best practice?

S1: "Pattern matching is more readable."

S1: "Here we use join to combine the values, we also tried  
zip and we think zip would work too"

OPE\_Bot: When done identifying the explanation, write the  
explanation agreed on in the chat.

S3: "We chose join to combine values and pattern  
matching because it is more readable"

OPE\_Bot: "Ok, we are switching roles now. Your next task  
is unlocked."

Figure 6.4: Example interaction in the collaborative reflection phase.

Task	Condition	Pretest(1)	Posttest(1)	Edges(4)	Vertices(5)	DF(6)	RDD(5)	PageRank (30)	Tuning (30)
1	Mostly problem-solving	0.5 (0.40)	0.55 (0.39)	3.63 (1.2)	4.54 (1.5)	5.45 (1.8)	4.54 (1.5)	24.5 (12.13)	24.5 (12.13)
	Equal	0.52 (0.36)	0.58 (0.39)	4 (0)	5 (0)	4.5 (2.66)	4.25 (1.83)	19.5 (14.68)	17.63 (14.8)
	Mostly worked ex. discussion	0.36 (0.39)	0.45 (0.35)	4 (0)	5 (0)	6 (0)	5 (0)	24 (12.65)	23.45 (12.47)
	Only worked ex. discussion	0.53 (0.43)	0.63 (0.45)	4 (0)	5 (0)	6 (0)	5 (0)	25.38 (11.27)	23.57 (12.12)
2	Mostly problem-solving	0.55 (0.28)	0.72 (0.36)	3.63 (1.2)	4.54 (1.5)	5.45 (1.8)	4.54 (1.5)	24.5 (12.13)	24.5 (12.13)
	Equal	0.63 (0.37)	0.77 (0.35)	4 (0)	5 (0)	4.5 (2.66)	4.25 (1.83)	19.5 (14.68)	17.63 (14.8)
	Mostly worked ex. discussion	0.59 (0.43)	0.72 (0.34)	4 (0)	5 (0)	6 (0)	5 (0)	24 (12.65)	23.45 (12.47)
	Only worked ex. discussion	0.59 (0.43)	0.9 (0.2)	4 (0)	5 (0)	6 (0)	5 (0)	25 (11.68)	23.04 (12.12)
3	Mostly problem-solving	0.75 (0.26)	0.88 (0.22)	3.63 (1.2)	4.54 (1.5)	5.45 (1.8)	4.54 (1.5)	24.5 (12.13)	24.5 (12.13)
	Equal	0.77 (0.25)	0.88 (0.27)	4 (0)	5 (0)	4.5 (2.66)	4.25 (1.83)	19.5 (14.68)	17.63 (14.8)
	Mostly worked ex. discussion	0.45 (0.47)	0.68 (0.33)	4 (0)	5 (0)	6 (0)	5 (0)	24 (12.65)	23.45 (12.47)
	Only worked ex. discussion	0.58 (0.36)	0.79 (0.29)	4 (0)	5 (0)	6 (0)	5 (0)	25.55 (10.86)	23.81 (11.37)
4	Mostly problem-solving	0.25 (0.35)	0.16 (0.35)	3.63 (1.2)	4.54 (1.5)	5.45 (1.8)	4.54 (1.5)	24.5 (12.13)	24.5 (12.13)
	Equal	0.47 (0.40)	0.52 (0.32)	4 (0)	5 (0)	4.5 (2.66)	4.25 (1.83)	19.5 (14.68)	17.63 (14.8)
	Mostly worked ex. discussion	0.36 (0.39)	0.45 (0.35)	4 (0)	5 (0)	6 (0)	5 (0)	24 (12.65)	23.45 (12.47)
	Only worked ex. discussion	0.58 (0.36)	0.79 (0.29)	4 (0)	5 (0)	6 (0)	5 (0)	25.55 (10.86)	23.81 (11.37)

Table 6.4: Verbose results showing means and standard deviations of the scores at different time points. Legend: Title Line: a (b) where 'a' represents the time point and 'b' represents the total possible score that can be obtained. Content Lines: i (j) where 'i' represent the mean scores of the students in that condition for that learning objective and 'j' represents the standard deviation.

### 6.2.3 Discussion

We find, interestingly, that the condition where students spent all of their time on collaborative reflection based on the presented worked example was best for conceptual learning from the task, and did not incur any deficit in terms of their ability to perform on the authentic task. The finding about conceptual learning impact is in line with the literature on example-based learning. For maximally impacting conceptual learning from the task, it is best to do away with superfluous production steps that might draw student attention away from conceptual learning, and focus all of their time instead on highlighting problem states relevant to conceptual learning. In the synchronous collaborative learning task that students participated in as a part of this study, this was achieved using a two-fold mechanism. In addition to simply being presented with a worked example, conversational agent prompts were used to direct student attention to the problem states relevant to their conceptual learning. In response to these prompts, students, in their groups were required to direct their attention to finding, and discussing the relevant problem states. The discussion meant that any misconceptions could be revealed and addressed for the group as a whole. An example of such an interaction in response to the collaborative reflection prompt is shown in Figure 6.4.

The finding about the lack of differential impact on the individual project is interesting. Since students are new to the procedural aspects of writing working programs in the new language, Scala, that is introduced in this unit, we hypothesized that hands-on practice would serve students better for performance on the authentic task. However, there turned out to not be a significant difference. When we asked students to understand why this might be the case, we found that most students were comfortable learning the syntactic and semantic aspects of programming in a new language all on

their own. Additionally, some foundational knowledge they had gained from working on other programming languages paradigms was useful in orienting them towards learning relevant syntactic and semantic aspects in this new language. Students preferred their time in the team programming project to be spent on learning concepts that they find harder to learn on their own. It should be noted, however, that the intervening week between the end of the collaborative activities and when the project was due could have served to wash out some of the differential impact. Since students said that they are comfortable searching for syntactic and semantic issues on their own, this could indeed have been the case. It is possible, however, that students who received less problem-solving practice spent more time searching, and less time on the implementation while those who participated in worked example study had to do more of the heavy-lifting around implementation themselves. This can only be confirmed by a process analysis of the data about how students split their time while working on the individual project. Since this study was not instrumented to collect this data, this is a caveat for these findings. We mitigate this concern by moving the individual project to immediately after the the team programming project in the subsequent study.

## 6.3 Studies 3 and 4 — Ordering and Time-Distribution between Problem-Solving and Worked Examples

For the third study, we repeat this experiment in the subsequent semester in the third project unit, where students have prior knowledge in the programming language used (Java). This time, we test all combinations of time spent on each treatment and the order of treatments, for a total of 8 conditions (4 treatments x 2 orders). For conceptual learning, presenting the worked examples first is better overall, and spending the entire time on the worked examples is the best individual condition. For individual project performance, problem-solving first is generally better. Some time problem-solving and most of the time on worked examples is the best individual condition. From studies 1 and 3, it is clear that maximizing time with worked examples is best for conceptual learning, and a little time on problem-solving is sufficient to help learners with prior procedural knowledge translate their conceptual understanding into computer programs.

For the final study, we repeat the experiment in the fourth project unit, where students do not have prior procedural knowledge in the programming language used (Scala). We also test the order of treatments this time, for a total of 8 conditions (4 treatments x 2 orders). Presenting worked examples first is better for conceptual learning, with mostly examples followed by some problem-solving being the best individual condition (though not significantly so). For individual project performance, problem-solving first is better overall, with half the time spent on each being the best individual condition.

Therefore, even when learners do not have prior procedural knowledge, maximizing time with worked examples is best for conceptual learning. They do need some additional time for practice, but spending more than half the time on problem-solving

is sub-optimal. The order of treatments has the same effect regardless of prior procedural knowledge — presenting worked examples first is better for conceptual learning, and problem-solving first is better for individual project performance. In either case, the findings support mostly worked example-based reflection, thus challenging the prevalence of problem-solving practice in today’s pedagogy of advanced computer science. The optimal design differs according to the learners’ prior procedural knowledge in the programming language used.

The results from all four experiments carried out in Chapter 6 are summarized in Table 6.5.

Table 6.5: Problem-Solving Practice versus Worked Examples for Different Levels of Prior Procedural Knowledge

#	Order of Treatments	Experimental Conditions	Number of Conditions	Programming Language	Prior Procedural Knowledge	Best for Conceptual Learning	Best for Individual Project
1 [146]	Problem-Solving First	Mostly Problem-Solving Mostly Worked Examples	2	Java	Present	Mostly Worked Examples	Mostly Worked Examples
2	Problem-Solving First Worked Examples First	Half the time Problem-Solving, then Examples Some Problem-Solving, then Examples Only Worked Examples Mostly Examples, then Problem-Solving Half the time on Examples, then Problem-Solving Some Examples, then Problem-Solving Only Problem-Solving Mostly Problem-Solving, then Examples Half the time Problem-Solving, then Examples Some Problem-Solving, then Examples Only Worked Examples Mostly Problem-Solving, then Examples Half the time Problem-Solving, then Examples Some Problem-Solving, then Examples Only Worked Examples Mostly Problem-Solving, then Examples Half the time on Examples, then Problem-Solving Some Examples, then Problem-Solving Only Problem-Solving	8 (4 Treatments x 2 Orders)	Java	Present	Worked Examples First; Only Worked Examples	Worked Examples First; Only Worked Examples
3 [147, 149]	Problem-Solving First		4	Scala	Absent	Only Worked Examples	No Significant Difference
4	Problem-Solving First Worked Examples First		8 (4 Treatments x 2 Orders)	Scala	Absent	Worked Examples First; Mostly Examples, then Problem-Solving	Problem-Solving First; Half the time Problem-Solving, then Examples

## 6.4 Chapter Discussion

It is clear that maximizing time with worked examples is best for conceptual learning, and a little time on problem-solving is sufficient to help learners with prior procedural knowledge translate their conceptual understanding into computer programs. Learners without prior procedural knowledge require additional time but it is still suboptimal to spend a majority of the time on problem-solving. The order of treatments has the same effect regardless of prior procedural knowledge — presenting worked examples first is better for conceptual learning, and problem-solving first is better for individual project performance. In either case, the findings support mostly worked example-based reflection, thus challenging the prevalence of problem-solving practice in today’s pedagogy of advanced computer science. The optimal design differs according to the learners’ prior procedural knowledge in the programming language used.

To deliver on the promise of effective programming projects, the ideal design needs to be sensitive to prior procedural knowledge levels. In the subsequent chapters, we will demonstrate two ways to do this. The first is a reinforcement learning (RL)-based strategy to determine the optimal duration split between the two treatments. The second modifies the task itself to interleave the effect of the two treatments. This interleaved design presents learners with two strategically different worked examples in pseudocode — an algorithmic description of the implementation. To choose a path to the solution, students engage in critical reflection, comparing conceptually relevant distinctions between worked examples. Then, in translating the pseudocode into computer programs, they receive problem-solving practice. We evaluate both strategies against baselines regardless of prior procedural knowledge levels to demonstrate their improved effectiveness.

# Chapter 7

## Reinforcement Learning-based Formulation for Adaptive Project Design

### 7.1 Introduction

In all of the studies presented thus far, in-process data from the experimental manipulations was analyzed only after the completion of all student interactions with the interventions. This data is used to test prevailing hypotheses and to understand the mechanisms through which individual student learning occurs within each experimental condition. Based on the results, hypotheses are refined and the interventions redesigned for future studies. Within a between-subjects experimental design, this data-driven approach facilitates the discovery of learning mechanisms that operate at the group level, on average.

If we conceptualize aspects of the intervention design as levers that can be manipulated to induce different behaviors, then the between-subjects design inherently limits the number of lever settings that can be explored at any given time. While significant differences in learning outcomes may emerge between conditions, it remains unclear if these reflect truly optimal lever configurations, due to —

- Combinatorial Complexity — Growth in possible combinations of design parameters.
- Context Sensitivity — Variability across learners or groups necessitating adaptive intervention strategies.
- Practical Constraints — The high cost and time demands associated with repeated factorial experiments.

To address these limitations, we draw upon concepts from Reinforcement Learning (RL) to frame the problem of adaptively searching the intervention parameter space. RL provides a principled framework for structuring the search for intervention parameters that optimize student learning and performance based on observed outcomes. Crucially,

while we employ an RL-inspired formulation to structure the problem, the lack of sufficient data in this context and existing domain knowledge about the parameter space led us to implement the adaptation using a specific heuristic resembling coordinate ascent. This means the policy for adjusting parameters was not learned from data, as in traditional RL. However, this same formulation is readily adaptable to policy learning via RL methods should sufficient data become available in other contexts, justifying the ‘RL-based’ conceptualization.

Unlike many prior applications of RL in education, which rely on simulations or offline data, our approach operates directly within a live collaborative learning environment. We constrain exploration using domain knowledge, employ episodic parameter updates based on batch rewards, and validate learning progress using proximal metrics that serve as reliable proxies for post-intervention outcomes. This design enables efficient, ethical, and context-sensitive adaptation of intervention parameters.

We review related work applying RL in educational contexts, highlighting how our use case diverges in both design and goals. This comparison serves to motivate the experimental innovations introduced in this chapter. We then frame the problem of selecting an appropriate parameter configuration for a conversational agent-based intervention as a parameter optimization problem structured using RL concepts.

A core innovation of our method lies in treating the reward surface over the parameter space as implicitly encoding transition dynamics — allowing for directional policy updates (parameter adjustments) without modeling explicit state transitions between fine-grained student interaction states. This idea aligns with model-free RL principles and provides a practical workaround for sparse or delayed reward scenarios common in education.

We test this novel approach in two naturalistic experiments. The first, a pilot study, assesses whether the framework can effectively identify optimal parameter settings that satisfy contextual constraints while improving proximal metrics correlated with learning outcomes. Following refinements based on the pilot, we conduct a second experiment demonstrating that the adaptive algorithm converges on an optimal parameter configuration within a single experimental cycle. The results show significant improvements in student learning and task performance compared to alternative settings, thereby validating the efficacy of our proposed approach.

## 7.2 Related Work

RL has seen limited application in educational settings, primarily due to its data-intensive nature. In typical RL formulations — such as those used for self-driving cars — agents interact freely with richly defined environments, requiring a vast number of iterations before converging on effective policies. These environments often employ a “self-play” paradigm, where the agent continuously engages with the environment in a game-like fashion to maximize a cumulative reward or “high score.” Learning effective policies under such conditions typically demands the use of Deep Reinforcement Learning (Deep RL), which integrates RL with deep neural networks to handle large,

unstructured state spaces that are not manually engineered.

In educational contexts, however, this paradigm introduces a unique ethical constraint — students cannot be knowingly subjected to detrimental interventions. To navigate this challenge, researchers have developed three broad categories of solutions that creatively adapt Deep RL methods for use in education.

The first category involves environments where student simulations can be constructed. These simulations approximate the “self-play” setup, allowing the RL agent to explore and learn policies without interacting directly with real learners. For example, Rafferty et al. [123] use a theoretically derived simulation to train an RL agent that selects instructional actions tailored to individual students. Similarly, Wang et al. [189] build simulations from prior student data to generate adaptive narratives in an educational game. However, these simulations are often limited by their inability to fully capture the complexities of real student behavior. As Doroudi et al. [46] note, discrepancies between simulated and actual learners can compromise the validity of such approaches.

This limitation becomes even more pronounced in collaborative learning contexts, such as ours, where the goal is to adapt the intervention based on group characteristics. In our case, the necessary theoretical models for simulating student groups do not yet exist — indeed, developing such models is one of the research goals. Moreover, despite access to several semesters’ worth of student interaction data, constructing realistic simulations remains infeasible. Even in individual learning scenarios, such simulations may require sophisticated models such as LSTMs to capture narrative sequences, as shown in Wang et al. [189]. Building accurate collaborative learning simulations represents an interesting but currently unattainable goal given the available data.

The second approach derives RL-based policies from pre-existing student interaction data, without constructing explicit simulations. In these cases, data limitations necessitate algorithmic adjustments to make learning feasible. For instance, Ausin [13] use Gaussian Processes to address the credit assignment problem by inferring immediate rewards from delayed outcomes. Other studies reduce the number of features considered to constrain the size of the state space [30], or incorporate empirically grounded assumptions to improve policy induction [32]. While such techniques may support effective policy learning, they are less suitable in our case where the intervention has evolved over time, making direct transfer from historical data unreliable.

The third approach applies RL in settings where large-scale student data can be gathered. For example, Reddy et al. [125] use crowdsourced data from the Duolingo platform to model optimal spaced repetition schedules. However, this method depends on high-volume, structured, and task-specific data, which is infeasible in our context. The nature of our intervention — embedded in a collaborative, naturalistic learning environment — cannot be adequately replicated in a crowdsourced setting. Thus, while such studies offer useful insights, the generalizability of their results to our use case remains limited.

Despite these constraints, prior work consistently shows that data-driven decisions about intervention design can significantly impact student learning [31]. In contexts where data is limited — as in our case — adaptive strategies inspired by RL principles

can still be viable. When the problem is carefully formulated with an informed design of the state space and reward structure, grounded in both domain knowledge and learning science, heuristic or optimization strategies guided by this formulation can produce meaningful educational benefits [47].

## 7.3 Methodological Innovations for Educational RL

Most prior experiments in applying RL to education recognize that interacting with real students is the ideal means of collecting data. However, they often avoid this approach due to data scarcity or ethical concerns. In contrast, we directly confront this challenge by proposing two key improvements.

First, we leverage our prior experience running conversational agent-based interventions in this context — as described in earlier chapters — to define a parameter space that constrains agent exploration to regions most likely to yield positive effects on student learning. This design helps mitigate ethical concerns by reducing the likelihood of testing highly suboptimal configurations during early adaptation phases.

Second, to address the data constraint, we introduce an Episodic Parameter Update methodology. Rather than making incremental updates after each granular interaction, we collect data across a full batch of group interactions (an “episode”) testing specific configurations of agent parameters. We then update the agent’s parameter configuration based on the average reward observed across the sessions within that episode. With a carefully selected step size and decay strategy, the adaptation heuristic converges towards an optimal parameter setting within a single experimental cycle. Figure 7.1 shows an overview of how we augment the Bazaar conversational agent framework — introduced in Chapter 4 — to support this adaptive parameter tuning. In this updated architecture, the design of the intervention is parameterized such that the optimization algorithm can adjust these parameters (e.g., prompt timing) to optimize toward externally defined success criteria (rewards). An automated analysis of session logs provides the feedback signal (reward) that drives the parameter updates.

In traditional experimental design, at least one full cycle is often dedicated to exploring the space of settings, with the best configuration deployed in a subsequent cycle. In contrast, our adaptive formulation allows an optimal configuration to be identified and increasingly utilized within the same cycle. Of course, this still raises ethical concerns — some students may experience suboptimal conditions during exploration. To mitigate this, we restrict the space of allowable configurations by eliminating known poor settings *a priori*, and the adaptive nature potentially exposes fewer students to suboptimal settings compared to a full factorial exploration. Arguably, similar ethical tradeoffs exist in traditional designs, but our approach provides the additional benefit of converging to an effective setting more efficiently.

In each episode (e.g., a day or a set number of groups), one or more parameter configurations are tested. Each configuration governs multiple collaborative sessions. The chat logs from sessions under a specific configuration are analyzed, and the resulting reward for that configuration is computed, typically by averaging across the sessions

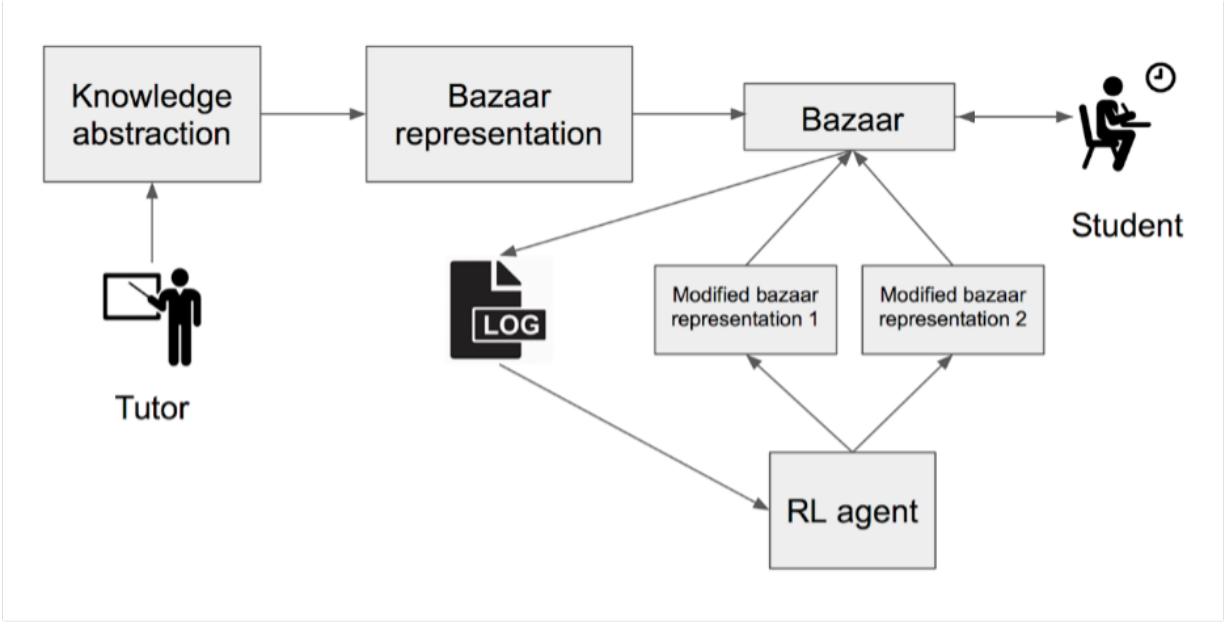


Figure 7.1: Augmenting the Bazaar Conversational Agent Framework for Adaptive Parameter Tuning

to handle variability. This average reward serves as the feedback signal for updating the strategy for choosing configurations in the next episode. By applying this process iteratively, the heuristic search converges toward a set of parameters that consistently yield high reward across episodes.

## 7.4 Parameter Optimization Problem Formulation

We frame the problem of adaptively tuning intervention parameters as an optimization task solvable with heuristic techniques guided by an RL-inspired formulation. While sharing core RL concepts like states, actions, and rewards, our approach focuses on iteratively refining a vector of parameters based on batch evaluations, rather than modeling step-by-step transitions within a single student session [97].

The problem can be defined by the following components —

- **Parameter Vector (State)  $t$ :** This vector represents the current configuration of the intervention parameters being optimized. In our pilot study,  $t = (t_1, t_2, t_3, t_4)$ , where  $t_i$  is the scheduled prompt timing in minutes for task  $i$ . The parameter space  $\mathcal{T}$  is the set of all possible valid vectors  $t$ .
- **Parameter Update (Action)  $a$ :** An action corresponds to selecting a modification or exploration strategy for the parameter vector. This might involve choosing which parameter(s)  $t_i$  to adjust and by what amount (e.g.,  $\pm\alpha$ , the step size), leading to a new candidate vector  $t'$  to be evaluated.
- **Reward Function  $R(t')$ :** This function provides a scalar evaluation of a specific

parameter configuration  $\mathbf{t}'$ . It is typically calculated as the average performance metric (e.g., proximal conversational score) observed across a batch of independent student sessions run using the configuration  $\mathbf{t}'$ .

- **Optimization Goal:** Find an optimal parameter vector  $\mathbf{t}^* \in \mathcal{T}$  that maximizes the expected reward  $E[R(\mathbf{t}^*)]$ .
- **Policy / Update Strategy  $\pi$ :** This is the algorithm used to decide which parameter configuration(s)  $\mathbf{t}'$  to evaluate in the next episode, based on the history of previously evaluated configurations  $\{\mathbf{t}_j\}$  and their observed rewards  $\{R(\mathbf{t}_j)\}$ . Our pilot study employed a strategy akin to coordinate ascent combined with a hill-climbing heuristic.

This formulation aligns with model-free RL concepts as it does not require an explicit model of how parameters influence the reward ( $R(\mathbf{t}')$  is treated as a black box). However, due to data limitations and existing knowledge about the likely shape of the reward surface, the update strategy  $\pi$  implemented in this work was a heuristic (coordinate ascent/hill-climbing) rather than a learned policy. This heuristic leverages the RL-inspired framing of states and rewards to guide the search efficiently. Importantly, this formulation could readily incorporate a learned policy  $\pi$  using standard RL algorithms if sufficient data were available for training, underscoring its potential as a general RL-based framework for intervention adaptation. Our episodic update process allows the algorithm to gather sufficient data for a reliable reward estimate  $R(\mathbf{t}')$  before making an update decision, addressing data sparsity and noise inherent in educational interactions. We use domain knowledge to constrain the parameter space  $\mathcal{T}$  and initialize the search, mitigating ethical concerns associated with exploring potentially detrimental configurations.

The specific heuristic determines how exploration (testing new configurations) and exploitation (using configurations known to yield high rewards) are balanced. For instance, a simple hill-climbing approach might involve testing configurations “neighboring” the current best  $\mathbf{t}$  (e.g.,  $\mathbf{t}$  with one  $t_i$  adjusted by  $\pm\alpha$ ) and moving to the neighbor if it yields a higher reward. The step size  $\alpha$  can be decayed over time to allow for finer tuning near optima.

## 7.5 Experiment - Pilot

### 7.5.1 Course Context and Experimental Design

The pilot experiment was conducted in the same context as the studies described in previous chapters — a graduate-level, project-based online course on Cloud Computing offered at Carnegie Mellon University. This pilot took place during the Spring 2020 offering of the course. As a reminder, the course structure is shown in Figure 7.2.

As before, students worked in groups of four on a collaborative programming project facilitated by a conversational agent. The project was divided into four tasks ( $k = 4$ ), each aligned with a specific learning objective. Each task lasted 14 minutes ( $T_{max} = 14$ )

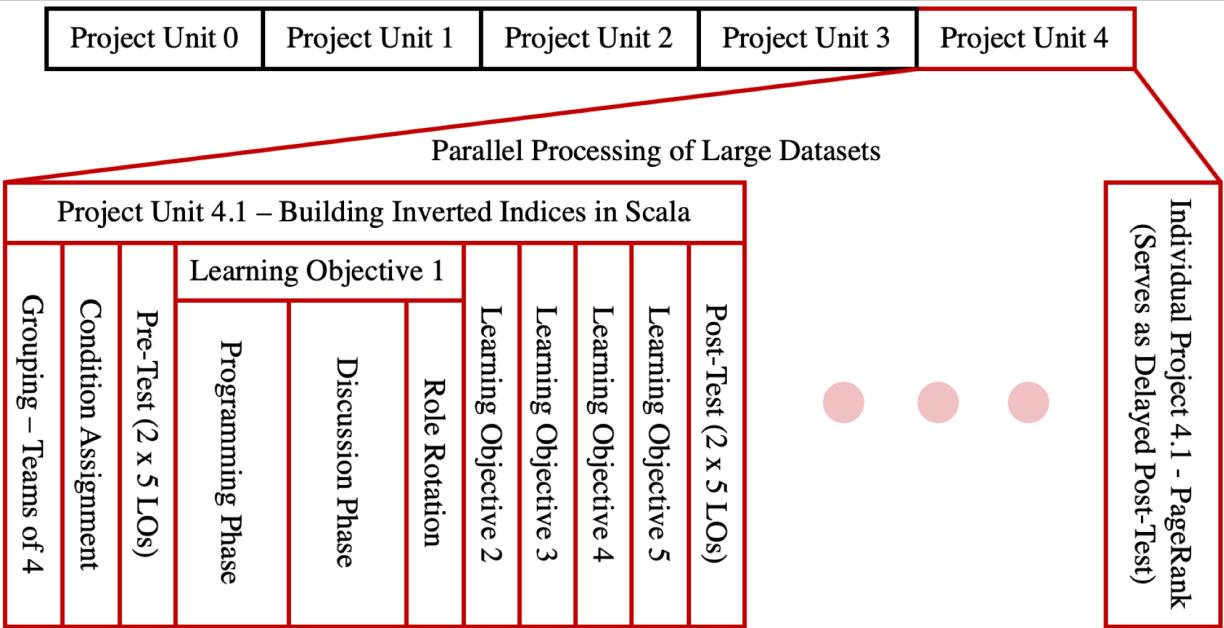


Figure 7.2: Course structure for the Cloud Computing course. The study reported in this chapter was conducted during Unit 4, which includes a collaborative project focused on constructing an inverted index in Scala.

and consisted of two phases — a problem-solving phase and a reflective discussion phase.

During the problem-solving phase, students were encouraged to debate competing implementation strategies. The agent played an active role by introducing prompts designed to surface conflicting design choices (conflict-oriented consensus building). The effectiveness of these prompts depended on timing. We therefore aimed to optimize the prompt delivery time  $t_i$  for each task  $i$  using an adaptive optimization approach.

Conflict-oriented consensus building is known to increase transactive exchange among group members and support deeper individual learning [194, 195, 203]. The constructive conflict resulting from students responding to these prompts is known to be one mechanism by which individual learning can happen from participating in a collaborative learning environment [195, 203].

Agent prompts were introduced with the specific intent to induce constructive conflict by proposing alternative implementation strategies. These prompts invited discussion and deliberation aimed at reaching consensus. However, the effectiveness of such prompts — whether they sparked engagement and led to productive resolution — was dependent on how far along the students were within the task. This highlights the importance of timing — the same prompt may be beneficial early in a task but disruptive if introduced too late.

Because the optimal timing of prompt delivery cannot be determined *a priori*, and likely varies across groups and tasks, we applied an adaptive approach to fine-tune prompt timing dynamically. Specifically, the agent adapted its policy for prompt de-

livery by learning from prior collaborative episodes, using the automatically detected presence of conflict-consensus dialogue as a proximal reward signal to guide the parameter adaptation.

### Conversational Indicators of Conflict-Consensus

Before the Unit 4 project where the experiment was conducted, students had participated in a prior collaborative project during Unit 3 of the course. This earlier project served as the source of training data for the supervised text classification model used in the experiment. The model was trained to assign a quality score to chat contributions, which was later used during the adaptation process as a reward signal.

An analysis of the Unit 3 chat logs indicated that agent prompts that elicited both conflict and consensus interactions were associated with greater student learning. To explore this relationship further, we conducted a qualitative comparison between student groups who demonstrated greater-than-expected learning gains and those who demonstrated lower-than-expected gains. These groups were identified using a median split on the residuals from a regression of post-test scores on pre-test scores. The high-gain groups exhibited richer, more interactive conversations, particularly with more frequent displays of conflict and consensus behaviors.

Based on these findings, we trained a supervised text classifier to distinguish chat contributions made by high-gain groups from those of low-gain groups, using features indicative of conflict and consensus. This classifier was then used during the Unit 4 experiment to automatically rate the quality of student discussions. These ratings served as a proxy for learning and were used as the reward function for the parameter adaptation algorithm.

As discussed earlier, this “proximal” measure of learning was necessary because the actual post-test and delayed post-test scores were not available until the conclusion of all collaborative episodes. While an alternative approach — such as the one used by Ausin [13], in which Gaussian Processes were used to infer immediate rewards from final outcomes — could have been employed to solve the credit assignment problem, our use of conversational indicators offers two advantages. First, these indicators are theoretically grounded and allow the parameter optimization algorithm to adapt using interpretable, pedagogically meaningful feedback. Second, this approach offers an opportunity to explore the relationship between these indicators and learning in this specific context, thus contributing back to learning science theory.

### Parameter Adaptation Process

The parameter adaptation process unfolded over the 5-day study duration, with different student groups participating in episodes. The goal was to find the optimal prompt timing vector  $\mathbf{t}^* = (t_1^*, t_2^*, t_3^*, t_4^*)$ . The adaptation heuristic adjusted the timing  $t_i$  for each task independently, resembling coordinate ascent combined with hill-climbing.

- Initialization — The process started with an expert-defined initial timing vector  $\mathbf{t}_0 = (6, 6, 6, 4)$ .

- **Exploration Strategy** — In each episode (e.g., a day’s worth of groups), for each task  $i$ , the heuristic typically tested two neighboring timing values around the current best estimate for  $t_i$ . Specifically, if the current timing was  $t_i$ , it might allocate some groups to test  $t'_i = t_i + \alpha$  and others to test  $t''_i = t_i - \alpha$ , where  $\alpha$  was the current step size. Approximately 3-5 groups ran in the experimental condition per episode, distributed across the tested parameter configurations.
- **Update Rule** — The average reward  $R(t'_i)$  and  $R(t''_i)$  (based on the classifier score for groups experiencing that timing) were compared. The timing for task  $i$  in the next episode,  $t_{i,next}$ , was moved in the direction that yielded higher reward. For example, if  $R(t_i + \alpha) > R(t_i - \alpha)$ , the next target timing for further exploration in that direction was  $t_i + 2\alpha$ . This assumes a locally monotonic reward surface.
- **Step Size Decay** — The step size  $\alpha$  started large (4 minutes, allowing broad exploration) and was reduced to 2 minutes once comparisons suggested a promising direction or region, enabling finer tuning. If exploration in one direction failed to improve the reward, the direction was reversed, with the reduced step size.
- **Handling Non-Presentation** — If groups finished a task before the scheduled prompt time  $t_i$ , preventing reward calculation, the heuristic interpreted this as needing an earlier prompt and adjusted  $t_i$  downward in the next episode.
- **Task Duration & Discretization** — Each task lasted 14 minutes. Timings  $t_i$  were chosen from discrete 2-minute intervals:  $\{0, 2, 4, 6, 8, 10, 12\}$ .

This iterative process of testing neighboring parameter values, comparing batch rewards, and moving towards higher-reward regions allowed the heuristic search to converge on effective timing parameters for each task.

## Mapping to MDP Concepts

While our approach employed a heuristic search rather than standard RL policy learning, mapping its components to a Markov Decision Process (MDP) framework can clarify the conceptual structure:

- **State space** — Each state  $s$  is the vector of prompt timings  $\mathbf{t} = (t_1, t_2, t_3, t_4)$ , where  $t_i \in \{0, 2, \dots, 12\}$ .
- **Action space  $A$**  — An action  $a$  selects which task coordinate(s)  $i$  to probe and potentially update. For instance, testing  $t_i \pm \alpha$ .
- **Transition function  $T$**  — Deterministic update  $T(\mathbf{t}'|\mathbf{t}, a)$  results from applying the chosen parameter adjustment  $a$  (e.g., setting  $t_{i,next} = t_i + \alpha$  if  $R(t_i + \alpha) > R(t_i - \alpha)$ ).
- **Reward function  $R(\mathbf{t}')$**  — Average classifier score for batches run with configuration  $\mathbf{t}'$ .
- **Discount factor  $\gamma$**  — Effectively  $\gamma = 1$ , as optimization is based on immediate batch rewards per episode.

Note again that the transition  $T$  and policy  $\pi$  were deterministically defined by the

coordinate ascent/hill-climbing heuristic in this implementation, not learned.

## Experimental Manipulation

In the control condition, students worked collaboratively without adaptive prompts during problem-solving, receiving only fixed discussion prompts afterwards (Table 7.1). A bottom-out hint was provided if needed.

Task	Prompt Text
Task 1	Discuss starting from the project manager — There are at least two ways to solve this task, using <code>reduceByKey()</code> or using <code>groupByKey()</code> along with other functions. What did you use and which one do you think works better in general?
Task 2	Project manager, please start this discussion now — Consider different ways of accessing the elements of a tuple within a <code>map()</code> function, e.g., dot-indexed or using the <code>case</code> function for pattern matching. What did you use and which do you think is better aligned with best practice?
Task 3	It's discussion time led by the project manager again — How did you choose to implement combining the values here? Can you unpack the working of your implementation?
Task 4	Final discussion time led by the project manager — Which RDD did you decide to cache? What factors did you need to keep in mind?
Follow-up Prompt	When done, write the explanation you have agreed on using the prompt "We chose ... because ..."

Table 7.1: Prompts presented in the discussion phase for control condition groups. The follow-up prompt is shown two minutes after the initial prompt and is the same across tasks.

The experimental condition included the same structure, but with an additional conflict-oriented prompt (Table 7.2) introduced during problem-solving at the time  $t_i$  determined by the adaptation algorithm for that episode. The content was matched to the control discussion prompts (Table 7.3) to isolate the effect of timing.

After each episode, chat logs were analyzed using the trained classifier. The average classifier score for groups experiencing a specific timing parameter  $t_i$  served as the reward signal  $R(t_i)$  to guide the adjustment of that parameter for the next episode via the described coordinate ascent / hill-climbing strategy.

Task	Prompt Text
Task 1	Discussing alternative implementations can help all of you learn from the task. Can you challenge the current implementation? I can think of a few ways of doing this task, like <code>reduceByKey()</code> or <code>groupByKey()</code> along with other functions. Why do you think the current implementation is better?
Task 2	What is the potential downside of your current implementation? For accessing array elements inside <code>map()</code> , for instance, I can think of dot indexing or using <code>case()</code> for pattern matching. Which do you think is better aligned with best practice?
Task 3	Combining values by key can be done in different ways. Would you consider implementing your own function or using an in-built one? Why or why not?
Task 4	Is caching all the RDDs acceptable? Why or why not?

Table 7.2: Conflict-oriented consensus-building prompts delivered during the problem-solving phase for experimental condition groups.

## 7.5.2 Results

The evaluation addressed two main questions — Did the adaptation strategy successfully converge and optimize for the target conversational behavior (conflict-consensus quality)? Did this optimization translate into increased student learning?

Regarding the first question, the adaptation process successfully converged on stable timing parameters by the fourth day of the five-day study. The sequence below illustrates the iterative refinement of the prompt timing vector  $\mathbf{t} = (t_1, t_2, t_3, t_4)$  across episodes (days), showing the parameter configurations and the logic driving the updates based on observed rewards and task progression —

**Episode 0 (Initial) —  $\mathbf{t} = [6, 6, 6, 4]$**   
(Expert-defined initial parameters)

**Episode 1 Update — Move to  $\mathbf{t} = [8, 8, 8, 2]$**

(Logic: For Tasks 1-3, exploration comparing timings with step size  $\alpha = 2$  favored later prompts  $[6 \rightarrow 8]$ . For Task 4, non-presentation triggered advancing the prompt  $[4 \rightarrow 2]$ .)

**Episode 2 Update — Move to  $\mathbf{t} = [12, 2, 2, 0]$**

(Logic T1: Direction  $6 \rightarrow 8$  was positive, continue exploring with larger step size  $\alpha = 4$   $[8 \rightarrow 12]$ .  
Logic T2/T3:  $t_{2/3} = 8$  performed worse than initial  $t_{2/3} = 6$ , reverse direction from 6 and explore with larger step size  $\alpha = 4$   $[6 \rightarrow 2]$ .  
Logic T4: Non-presentation again, advance prompt  $[2 \rightarrow 0]$ .)

**Episode 3 Update — Move to  $\mathbf{t} = [10, 0, 0, 0]$**

Task	Prompt Text
Task 1	I mentioned a couple of ways of solving the problem earlier, like using <code>reduceByKey()</code> or <code>groupByKey()</code> along with other functions. Which of these did you use and which is better in general?
Task 2	I highlighted a few different ways of doing element indexing while you were working on the task, such as dot indexing or the Scala <code>case()</code> function. Which of these did you use, and which one would be better aligned with best practices?
Task 3	We talked about either implementing your own function or using an in-built one for combining the values here. Which one did you choose? Please unpack the working of your implementation.
Task 4	Which RDD did you choose to cache? What factors did you need to keep in mind while making the decision?
Follow-up Prompt	When done, write the explanation you have agreed on using the prompt "We chose ... because ..."

Table 7.3: Prompts presented in the discussion phase for experimental condition groups. The follow-up prompt is shown two minutes after the initial prompt and is identical across tasks.

(Logic T1:  $t_1 = 12$  worse than  $t_1 = 8$ , step back with decayed step size  $\alpha = 2$  [ $12 \rightarrow 10$ ] and converge.

Logic T2/T3:  $t_{2/3} = 2$  performed well, continue exploring downwards, clamped at boundary [ $2 \rightarrow 0$ ].

Logic T4: Already converged at  $t_4 = 0$ .)

**Episode 4 Update / Final Converged Parameters —  $\mathbf{t}^* = [10, 2, 2, 0]$**

(Logic T1/T4: No change, already converged.

Logic T2/T3:  $t_{2/3} = 0$  performed worse than  $t_{2/3} = 2$ , step back with decayed step size  $\alpha = 2$  [ $0 \rightarrow 2$ ] and converge.)

Next, we assessed correspondence between the optimized parameters and actual behavior using post-hoc analysis of annotated conflict/consensus instances. The annotation scheme was —

- Contribution introducing new idea = 1 (conflict).
- Response aiming to build consensus = 1 (consensus).
- Agent guidance = 1 (conflict-oriented consensus).
- Consecutive/repeated statements handled appropriately.

Two researchers independently annotated the chat contributions. Inter-rater reliability was high (Cohen's  $\kappa = 0.87$ , indicating substantial agreement). We observed that the

adaptation process, driven by the classifier reward, indeed led to configurations that increased the total number of conflict and consensus-building attempts compared to initial settings. Thus, the adaptation strategy successfully optimized for parameters enhancing the target conversational behaviors.

Regarding the second question, we evaluated the correlation between these conversational behaviors and learning outcomes (pre-post test gains). The total number of conflict-oriented consensus-building attempts (the initial target proxy) was significantly correlated with learning for only one task, marginally for another, and uncorrelated for two. This suggests optimizing solely for this combined metric did not consistently improve learning.

Further post-hoc analysis separating conflict and consensus attempts revealed that consensus-building attempts were consistently correlated with learning across all tasks, while conflict attempts were not. This contrasts with earlier observations (Unit 3) where conflict seemed more directly linked. It suggests conflict is beneficial only when resolved through productive consensus; optimizing for conflict alone can be detrimental if it doesn't lead to integration.

Therefore, we refine our hypothesis — consensus-building, perhaps scaffolded by conflict-inducing prompts, is the more crucial driver of learning in this context. Based on this, we propose a follow-up study where the adaptation algorithm optimizes parameters specifically to maximize consensus-building attempts.

### 7.5.3 Discussion

While the pilot successfully demonstrated the framework's ability to adapt intervention parameters towards desired conversational patterns, a key limitation was that optimizing the initial proxy reward did not directly translate into improved learning outcomes in this iteration. However, the study yielded a refined, data-driven understanding of the interplay between conflict and consensus in collaborative learning, informing both learning science theory and the design of the follow-up experiment aimed at demonstrating downstream learning gains. Example interactions with productive integration as well as unresolved conflict, both triggered by prompts introduced by the conversational agent are shown below

*Scenario: The group is implementing Task 1 (inverted index) using groupByKey. The agent sends the conflict-oriented prompt (Table 5, Task 1) to the Project Manager.*

#### Example 1 — Conflict Leading to Consensus

*Scenario: The group is implementing Task 1 (inverted index) using groupByKey. The agent sends the conflict-oriented prompt (Table 5, Task 1) to the Project Manager.*

**Agent (to Project Manager):** Discussing alternative implementations can help all of you learn from the task. Can you challenge the current implementation? I can think of a few ways of doing this task, like reduceByKey() or

groupByKey() along with other functions. Why do you think the current implementation is better?

**Project Manager:** Hey team, the agent just raised a good point. We're using groupByKey right now after mapping words to (docID, 1) pairs, right? The agent mentioned reduceByKey as an alternative. Should we consider that instead? Why did we choose groupByKey? (Annotation: *Conflict initiated*)

**Driver:** I went with groupByKey because it felt straightforward – group all the counts for each word together, then sum them up in the mapValues step. It directly gives us the list we need for the index eventually. (Annotation: *Consensus-building - responding to query*)

**Researcher:** Hmm, I remember reading that groupByKey can be inefficient. It shuffles *\*all\** the values for a key to one node before combining. If a word is really common, that could be a lot of data transfer and potentially cause memory issues. (Annotation: *Conflict reinforced - potential issue raised*)

**Navigator:** That's a good point, Researcher. So reduceByKey would be better how? Does it combine things earlier? (Annotation: *Consensus-building - seeking understanding*)

**Researcher:** Yeah, reduceByKey applies the combining function (like summing the counts) *\*before\** shuffling data across the network, or at least partially combines locally first. So, less data gets shuffled. For simple sums like word counts, it's usually preferred over groupByKey. (Annotation: *Consensus-building - providing information for comparison*)

**Project Manager:** Okay, so reduceByKey seems more efficient, especially for large datasets. Driver, would it be hard to switch? We'd need to emit (word, count) pairs and then just use reduceByKey(\_ + \_)? (Annotation: *Consensus-building - integrating information, proposing path forward*)

**Driver:** No, that should be easy enough to change. Instead of mapping to ((word, docId), 1) then grouping by word, we could map to (word, 1) and use reduceByKey directly for counts per word first... wait, we need counts *\*per document\**, right? So maybe map(word => ((word, docId), 1)).reduceByKey(\_ + \_)? That gives us ((word, docId), count). Then we'd still need to group by word later to get the list of docIDs. Is reduceByKey still better then? (Annotation: *Conflict / Consensus-building - refining understanding, identifying complication*)

**Navigator:** Good catch. So maybe the initial groupByKey *\*after\** getting (word, docId) pairs is okay if we need the list of documents anyway? Or maybe there's a way to use reduceByKey cleverly... Let's stick with the reduceByKey approach for counts per doc first: map(line => ... ((word, docId), 1)).reduceByKey(\_ + \_). Then we have RDD[((word, docId), count)]. From there, we can

`map { case ((word, docId), count) => (word, (docId, count)) }` and \*then\* maybe use `groupByKey` just to collect the `(docId, count)` pairs for each word? That might shuffle less data overall than grouping all the initial `(docId, 1)` pairs. (Annotation: *Consensus-building - synthesizing and proposing a concrete plan*)

**Project Manager:** Okay, that sounds like a reasonable plan that uses `reduceByKey` for the heavy lifting first. Let's try that approach. Driver, can you implement that sequence? (Annotation: *Consensus - reaching agreement*)

**Driver:** Yep, got it. Switching now. (Annotation: *Consensus - action based on agreement*)

## Example 2 — Conflict without Consensus

*Scenario: The group is implementing Task 1 (inverted index) using `groupByKey`. The agent sends the conflict-oriented prompt (Table 5, Task 1) to the Project Manager.*

**Agent** (to Project Manager): Discussing alternative implementations can help all of you learn from the task. Can you challenge the current implementation? I can think of a few ways of doing this task, like `reduceByKey()` or `groupByKey()` along with other functions. Why do you think the current implementation is better?

**Project Manager:** Okay team, the agent is asking about our use of `groupByKey`. It mentioned `reduceByKey` as another option. Any strong feelings why we should stick with `groupByKey`? (Annotation: *Conflict initiated*)

**Researcher:** Well, I know `groupByKey` can be risky for performance if one key has tons of values. It shuffles everything first. `reduceByKey` avoids that for operations like summing counts. (Annotation: *Conflict reinforced - potential issue raised*)

**Driver:** I already started coding it with `groupByKey`. It makes sense to me to get all the documents for a word together. Switching now will slow us down, and I doubt the dataset is big enough here to actually cause memory problems. Let's just finish it this way. (Annotation: *Resistance / Dismissal of alternative*)

**Navigator:** Uh, maybe the researcher has a point about efficiency later... but getting it done is important too. Driver, if you're already on track, just keep going with `groupByKey` for now. We can flag it as a potential optimization if needed. (Annotation: *Decision made without integration of proposed conflict / Deferral*)

**Project Manager:** Alright then. Let's proceed.

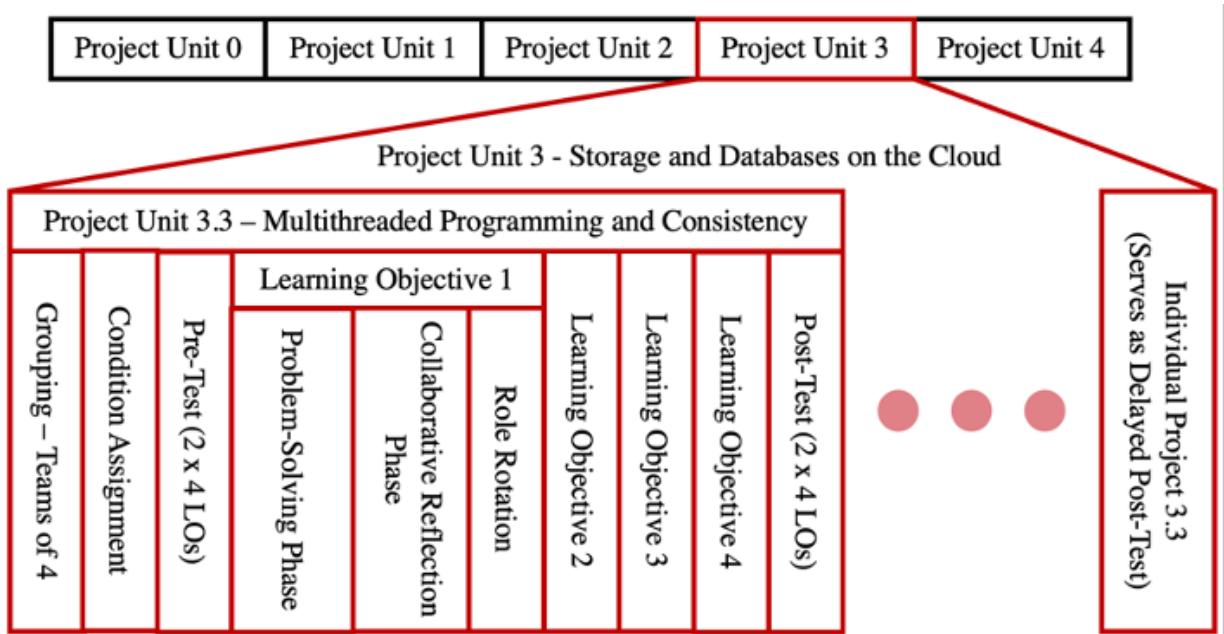


Figure 7.3: Course structure for the Cloud Computing course. The deployment study was conducted in Unit 3, which includes a collaborative project focused on multi-threaded programming and consistency.

## 7.6 Deployment Experiment

### 7.6.1 Course Context and Experimental Design

This follow-up study was conducted to apply the refined understanding from the pilot and demonstrate the efficacy of the adaptive parameter optimization approach in producing downstream learning gains within a larger experimental context. The study took place during Unit 3 of the Spring 2021 offering of the Cloud Computing course (see Figure 7.3 for course context). Using Unit 3 data allowed the text rating model (trained on data from a previous offering’s Unit 3) to operate in the domain it was trained on, potentially increasing the accuracy of the proximal reward signal and reducing uncertainty from possible misattributions compared to the pilot in Unit 4.

In this study, the intervention component optimized via adaptive parameter tuning was a prompt delivered to the group’s project manager during the programming phase. This prompt encouraged the manager to initiate a brief code review, stepping back to suggest possible alternative implementations to the group. The primary goal of the adaptive tuning component was to determine the optimal timing for this project manager prompt to maximize positive conversational indicators (ideally, consensus-building, informed by the pilot study findings).

Due to the number of students available (117 students, forming 32 groups), the state space for the parameter optimization (the possible timings for the project manager prompt) was reduced compared to the pilot. It consisted of three discrete configura-

tions — delivering the prompt within the First third of the task duration, around the Half-way point, or within the Last third of the task.

## Experimental Manipulation

This adaptive optimization was embedded within a larger 2x2 factorial experimental design, allowing us to investigate both the main effects of the broader intervention structure and the optimized timing within those structures. The primary factors of the 2x2 design were —

- **Sequence** — Programming Practice phase first followed by Collaborative Reflection phase second, versus Collaborative Reflection first followed by Programming Practice second. Time was split equally (50-50) between phases in both sequence conditions.
- **Solution-Path Prompting** — During the Collaborative Reflection phase (regardless of whether it came first or second), the agent prompts focused on comparing the group's approach to either a conceptually *Similar* alternative or a conceptually *Different* alternative.

This 2x2 design resulted in four main experimental conditions, with 8 groups assigned to each condition (total 32 groups). Within each of these four conditions, the adaptation algorithm was employed independently to determine the optimal timing for the project manager code review prompt during the programming phase. An overview of this design is presented in Table 7.4.

Sequence	Solution-Path Prompting		Total Groups
	Similar	Different	
Programming Practice (1st) → Collab. Reflection (2nd)	Condition 1 (N = 8 groups)	Condition 2 (N = 8 groups)	16
Collab. Reflection (1st) → Programming Practice (2nd)	Condition 3 (N = 8 groups)	Condition 4 (N = 8 groups)	16
<b>Total Groups</b>	16	16	<b>32</b>

Table 7.4: Overview of the 2x2 Factorial Experimental Design (Deployment Study, Spring 2021 Unit 3). Note: Adaptive optimization of Project Manager prompt timing was conducted within each of the four conditions (C1-C4). Total participants: 117 students.

The parameter adaptation process within each condition followed an episodic update strategy, unfolding over the duration of the experiment as groups participated. The process involved distinct phases —

- **Exploration Phase 1** — Tested the effectiveness of delivering the project manager prompt in the *First 1/3rd* of the task.

- **Exploration Phase 2** — Tested the effectiveness of delivering the prompt in the *Last 1/3rd* of the task.
- **Exploitation Phase** — Based on reward signals (classifier scores) from the exploration phases, tested the timing configuration that appeared most promising or explored the remaining option (*Half Way*). ‘Half Way’ was tested here, potentially if neither extreme was clearly superior initially, or as a comparison point.
- **Converged Phase** — Deployed the timing configuration identified as optimal based on the accumulated reward data from the previous phases.

As groups participated in the project over several time slots, the heuristic gathered reward data for the different timings within each of the four main conditions. The automated analysis of chat logs provided the reward signal (proximal conversational quality score) after each batch of sessions, informing the parameter settings used in subsequent sessions. Table 7.5 provides a schematic overview of how groups were allocated to different adaptation phases (Exploration, Exploitation, Converged) across time slots within the main experimental conditions.

Table 7.5: Detailed Schedule of Parameter Adaptation for Project Manager Prompt Timing (Deployment Study)

Condition Cell from Factorial Design	Groups	Adaptation Phase	Prompt Timing
P→R / Sim	2	Exploration	First 1/3rd
P→R / Sim	2	Exploration	Last 1/3rd
P→R / Sim	2	Exploitation	Half Way
P→R / Diff	2	Exploration	First 1/3rd
P→R / Diff	2	Exploration	Last 1/3rd
R→P / Sim	2	Exploration	First 1/3rd
R→P / Sim	2	Exploration	Last 1/3rd
R→P / Diff	2	Exploration	First 1/3rd
R→P / Diff	2	Exploration	Last 1/3rd
P→R / Diff	2	Exploitation	Half Way
R→P / Sim	2	Exploitation	First 1/3rd
R→P / Diff	2	Exploitation	First 1/3rd
P→R / Sim	2	Converged	First 1/3rd
P→R / Diff	2	Converged	First 1/3rd
R→P / Sim	2	Converged	First 1/3rd
R→P / Diff	2	Converged	First 1/3rd

*Note:* Condition Cell abbreviations: P→R = Programming Practice (1st) → Collab. Reflection (2nd); R→P = Collab. Reflection (1st) → Programming Practice (2nd); Sim = Similar Solution-Path Prompting; Diff = Different Solution-Path Prompting. Across all four main experimental conditions, the adaptation algorithm converged on the same optimal timing for the

project manager code review prompt — delivering it within the First third of the programming task phase. This suggests that, regardless of the overall sequence or the type of solution-path prompting used in reflection, initiating the project manager’s code review prompt relatively early in the programming phase was most effective at promoting the desired conversational dynamics (as measured by the reward signal). This indicates a stable outcome from the optimization process. The subsequent sections will analyze the impact of both the main experimental conditions and the converged parameter on learning outcomes.

### 7.6.2 Analysis and Results

This section focuses on the results related to the adaptive tuning of the project manager prompt timing within the deployment study. The analysis addresses three key questions —Did the adaptation process converge on an optimal timing parameter? Did the converged timing parameter lead to an increase in the desired proximal conversational behavior (consensus-building)? Did optimizing for this proximal behavior translate into improvements in distal learning outcomes (conceptual learning and transfer task performance)?

#### Convergence of Parameter Adaptation

As detailed in the experimental design and illustrated in the adaptation schedule (Table 7.5), the adaptation algorithm responsible for tuning the project manager prompt timing successfully converged. Despite operating independently within the four main experimental conditions derived from the 2x2 factorial design, the algorithm consistently identified delivering the prompt within the First 1/3rd of the programming task phase as the optimal timing across all conditions by the end of the study period. This indicates a stable outcome from the optimization process.

#### Impact on Consensus Behavior

To verify that the converged parameter setting enhanced the targeted conversational behavior, we analyzed the chat logs for instances of consensus-building attempts, drawing on the refined understanding from the pilot study. Following a similar annotation procedure as the pilot (inter-rater reliability remained high with  $\kappa = 0.85$ ), we counted the frequency of consensus-building contributions within each group’s programming phase.

We conducted a one-way ANOVA comparing the average number of consensus-building attempts across the three timing configurations tested during the experiment (*First 1/3rd*, *Half Way*, *Last 1/3rd*), pooling data across the four main experimental conditions as no significant interaction between the main conditions and timing effect on consensus was found. The analysis revealed a significant effect of the prompt timing on consensus-building frequency,  $F(2, 29) = 5.72, p < 0.01$ . Post-hoc comparisons indicated that groups receiving the prompt in the *First 1/3rd* (the converged setting) exhib-

ited significantly more consensus-building attempts (Mean = 7.5, SD = 2.0) compared to both the *Half Way* (Mean = 5.0, SD = 1.8,  $p < 0.05$ ) and *Last 1/3rd* timings (Mean = 4.0, SD = 1.5,  $p < 0.01$ ). This confirms that the parameter optimization algorithm successfully optimized for a timing that increased the desired proximal conversational behavior.

### Impact on Learning Outcomes

We next investigated whether this increase in consensus behavior, facilitated by the converged timing, translated into downstream learning benefits.

Conceptual learning was measured by gains from the pre-test to the post-test administered immediately before and after the collaborative project. We performed an ANCOVA predicting post-test scores, with the converged prompt timing configuration (*First 1/3rd* vs. *Half Way* vs. *Last 1/3rd*) as the independent variable, controlling for pre-test scores. The results showed a significant main effect of the timing configuration on conceptual learning gains,  $F(2, 113) = 4.15, p < 0.05$  (assuming student-level analysis,  $df_2 \approx N_{\text{students}} - N_{\text{groups}}$  or  $N_{\text{students}} - N_{\text{params}}$ ). Specifically, students in groups that experienced the converged *First 1/3rd* prompt timing demonstrated significantly higher adjusted post-test scores (indicating greater conceptual learning) compared to those experiencing the *Last 1/3rd* timing ( $p < 0.05$ ). The comparison with the *Half Way* timing approached significance ( $p = 0.07$ ). This suggests that optimizing for consensus-building via prompt timing positively impacted conceptual understanding gained from the collaborative task. This alignment between optimizing for the proximal conversational metric (consensus) and achieving downstream conceptual learning gains supports the validity of the chosen reward signal and the underlying assumption about the utility of early prompting in this context.

Performance on the subsequent individual programming project served as the measure of problem-solving transfer. We conducted a similar ANCOVA predicting the individual project score, with prompt timing configuration as the factor and pre-test score (or another relevant covariate like prior programming experience) included to control for prior differences.

The analysis revealed no significant detrimental effect of the prompt timing manipulation on transfer performance,  $F(2, 113) = 0.88, p = 0.42$  ( $p > 0.05$ ). Groups experiencing the converged *First 1/3rd* timing did not perform worse than other groups. Furthermore, while not statistically significant, there was a positive trend indicating potentially improved performance for the *First 1/3rd* timing group (Mean Score = 90.5) compared to the other timing groups combined (Mean Score = 88.5). This suggests that the parameter optimized timing successfully enhanced proximal behaviors and conceptual learning without hindering, and possibly slightly benefiting, students' ability to apply their knowledge on a future task.

### 7.6.3 Discussion

The adaptive algorithm successfully converged on delivering the project manager prompt in the *First 1/3rd* of the task as the optimal timing across different experimental con-

ditions. This converged setting was empirically verified to significantly increase the frequency of consensus-building behaviors compared to other timings. Crucially, this optimization for proximal conversational indicators translated into tangible learning benefits, with the converged timing leading to significantly greater conceptual learning gains. Moreover, this was achieved without any detriment to performance on a delayed transfer task, with trends even suggesting a slight improvement. These findings validate the use of this adaptive parameter optimization approach for enhancing collaborative learning interventions.

## 7.7 Chapter Discussion

This chapter introduced and evaluated an adaptive approach, formulated using Reinforcement Learning (RL) principles, for optimizing intervention parameters within live, collaborative learning environments. Moving beyond traditional between-subjects designs, we aimed to develop a methodology capable of efficiently searching the parameter space to enhance educational support under real-world constraints. The deployment experiment, building on insights from the pilot study, successfully demonstrated the efficacy of this approach. The adaptation algorithm converged on an optimal timing for the project manager code review prompt, which demonstrably increased consensus-building behaviors among students and, crucially, led to significant improvements in conceptual learning without hindering performance on a subsequent transfer task.

A key motivation for employing a formulation inspired by RL stemmed from the varying learner characteristics across these learning environments. Specifically, the pilot study was situated in Unit 4, where students encountered the material with little prior procedural knowledge in the programming language used for the implementation. In prior studies, we established that this necessitates more time dedicated to problem-solving, making the timing of reflective prompts particularly critical — introduced too early, they might be irrelevant; too late, and they might disrupt crucial problem-solving momentum. The parameter optimization algorithm in the pilot was tasked with navigating this sensitive timing issue for conflict-oriented prompts. While it successfully optimized for conversational indicators, the link to learning was complex, revealing that maximizing conflict alone was insufficient.

The deployment experiment shifted to Unit 3, a context where students generally possessed prior procedural knowledge. Even here, however, optimizing intervention timing remains important for effective scaffolding since some amount of problem-solving is still necessary, as established in Chapter 6. We adapted the approach based on the pilot’s findings, focusing on optimizing the timing of a project manager prompt using consensus-building indicators (implicitly, via the classifier trained on high-gain groups exhibiting both conflict and consensus resolution). The adaptation algorithm efficiently determined that delivering this prompt relatively early (First 1/3rd of the task) was optimal across different base experimental conditions (variations in sequence and solution-path prompting). This convergence provides a concrete, validated parameter setting that can be deployed in future iterations of this Unit 3 project, yielding consistent learn-

ing benefits.

Beyond this specific finding, our work highlights several factors critical for successfully applying RL-inspired adaptive techniques in practical educational settings, where large-scale data collection typical of other RL domains (like games or simulations) is often infeasible or unethical —

- **Data Scarcity and Episodic Updates** — Educational interactions yield data slowly compared to simulations. Our episodic update strategy, averaging rewards over batches of groups experiencing the same parameter configuration, provides a robust signal for adaptation despite noisy, limited data per interaction.
- **Constrained Parameter Space** — The number of participating learners directly limits the complexity of the parameter space that can be effectively explored within a reasonable timeframe (e.g., a single experimental study). We addressed this by focusing on optimizing a small number of critical parameters (primarily timing) and discretizing the state space (e.g., First 1/3rd, Half Way, Last 1/3rd). Domain knowledge and prior studies are essential for identifying which parameters are likely to have the most impact and warrant optimization.
- **State Space Assumptions and Heuristics:** Making reasonable assumptions about the reward landscape — such as local monotonicity near the optimum — allows for the design of efficient exploration heuristics like the coordinate ascent/hill-climbing strategy used here. This heuristic leverages the RL-inspired framing to guide the search effectively without exhaustive exploration or explicit policy learning.
- **Theory-Grounded Proximal Rewards** — When distal learning outcomes (e.g., final project scores, post-test gains) are delayed, using theoretically grounded proximal indicators from intermediate data (like specific conversational patterns correlated with learning) is required. The success hinges on the validity of these proxies, as demonstrated by the refinement needed after the pilot study (shifting focus from raw conflict to consensus). Validating these proxies remains an important step, but can contribute to improving our understanding of the learning process from learning science theory.
- **Ethical Considerations** — Starting with expert-defined reasonable parameters and constraining exploration helps mitigate the risk of exposing students to significantly detrimental conditions during the adaptation process.

In essence, this work demonstrates a practical methodology for leveraging RL concepts for in vivo tuning of educational interventions. It allows instructional designers and researchers to move beyond static designs towards adaptive systems that optimize support based on observed effectiveness within the deployment context itself. While the heuristic search used here is simpler than policy learning algorithms often employed in simulation-based educational RL, they prove effective when combined with pedagogical knowledge, careful problem formulation, and realistic constraints. Importantly, the underlying RL-inspired formulation provides a clear pathway for incorporating more sophisticated, data-driven policy learning algorithms should sufficient data be-

come available in future applications.

Future work could explore optimizing multiple parameters simultaneously (requiring more sophisticated algorithms or larger student cohorts), investigating more complex state representations that incorporate real-time group dynamics, and validating the approach across different course contexts and intervention types. Nonetheless, the current findings represent a significant step towards making data-driven, adaptive instructional design a practical reality in authentic learning environments.



# Chapter 8

## Interleaved Worked Examples and Problem-Solving Practice

Prior studies employed a “phased” design, common in scripted collaboration [53], to divide time between worked example-based reflection and problem-solving implementation phases using a macroscript. The boundary between these phases was varied in a between-subjects design to examine the differential impact of each on learning outcomes. The results indicated that worked example-based reflection was more effective for conceptual learning, challenging the status quo in CS education pedagogy. While these findings were informative, it remains unclear whether the phased design represents the most effective use of learners’ time. A pertinent question is whether it is possible to enhance the benefits by increasing exposure to reflection opportunities without negatively impacting problem-solving transfer.

One way to increase exposure while keeping total time-on-task constant is by embedding reflection opportunities “in the flow” of problem-solving implementation rather than separating them into distinct phases. This “interleaved” design integrates reflection prompts throughout the task — potentially increasing their salience. However, it may also divide students’ attention, lacking the clear separation provided in a phased design. The idea of reflection during problem-solving practice has precedent in cognitive science, particularly with the “self-explanation effect,” where individuals benefit from explaining their reasoning during problem-solving implementation [7] — often while studying worked examples [36]. While these studies were conducted in individual contexts, it is reasonable to hypothesize that collaborative reflection may play a similar role in group settings. The mechanisms underlying learning from one’s own reasoning and from the articulated reasoning of peers in collaborative learning environments are discussed in Chapter 2. To evaluate this hypothesis, we compare a design that prompts collaborative reflection “in the flow” of problem-solving implementation with the traditional phased design.

To create a project that integrates worked example-based reflection with ongoing problem-solving practice, we draw inspiration from tasks such as Parsons Problems in CS education [49], which blend reflection and problem-solving elements, and adapt them for collaborative use. In our novel project design, each group is presented with

two alternative worked example solutions for each task in the project, written as pseudocode. The solutions are constructed to draw the joint attention of the group to the pertinent conceptual aspects that they are expected to learn. Conversational agent-based prompts support the group in this decision-making process. Throughout, students are free to implement their chosen solution, moving fluidly between reflection and implementation as needed [150].

In contrast, the phased design presents only a single worked example solution during the reflection phase. Instead of an alternative example, it is the conversational agent that introduces conceptually relevant aspects of an alternative solution for the group to consider. After reflection, the group transitions to a problem-solving implementation phase to implement their selected solution. In both conditions, the implementation requires translating the pseudocode into a working program. While the interleaved and phased designs differ structurally, both present the same conceptual content to learners, ensuring information was controlled across conditions; both groups received the same core conceptual points and overall worked example logic, differing only in the timing and format of presentation (single vs. paired examples within distinct phases vs. integrated).

This study tests the hypothesis that interleaving worked example-based reflection with problem-solving implementation enhances outcomes beyond the phased approach by increasing reflection opportunities while maintaining total time-on-task. In a between-subjects design, the interleaved and phased approaches are compared based on conceptual learning (measured via pre- to post-test gains) and performance on subsequent individual programming projects, which serve as measures of problem-solving transfer.

Results show that the interleaved approach maintains the high level of conceptual learning achieved by the phased approach while improving performance on individual programming projects. Thus, the novel interleaved design amplifies the benefits to future problem-solving transfer. The following sections describe the experimental methods, including the study context, the overall collaborative project design, participant and group composition, and the between-subjects conditions. We then present our analysis, key results, implications, and future directions.

## 8.1 Course Context and Experimental Design

This experimental study was conducted in the Summer 2021 offering of a graduate-level, semester-long online course titled “Foundations of Computational Data Science.” The course introduces advanced data science concepts and is offered to graduate students in the Masters in Computational Data Science program at Carnegie Mellon University. As such, it functions as an advanced course within the computer science curriculum. This setting provided a context with motivated students possessing prerequisite programming skills, allowing the study to focus on the collaborative and learning process aspects rather than basic programming challenges.

Similar to the Cloud Computing course, this course is structured around seven project-based units, each culminating in an individual programming project with assessment

components for the unit. This experiment was situated within the seventh project unit, which focuses on evaluating and optimizing machine learning models. As preparation for the individual programming project in this unit, students worked in groups of four on a synchronous collaborative programming project called the Online Programming Project (OPP). For the project, students used a collaborative programming environment (Cloud9), instrumented with a text-based chat tool for communication. A conversational agent, built using the Bazaar framework [3], was embedded in the chat to provide project instructions and facilitate both worked example-based reflection and problem-solving implementation.

The project lasted 80 minutes and was divided into four tasks, each lasting 20 minutes and targeting a distinct learning objective (LO). Depending on the experimental condition assigned to the group, each task followed either a phased design (with separate collaborative reflection and implementation phases) or an interleaved design (with both processes occurring together). In either case, a role scaffold was used to structure the collaboration by assigning the four students to complementary roles, designed to distribute cognitive load and encourage active participation from all members. This scaffold builds on prior work adapting the Mob Programming paradigm from industry [199, 204] for use in online instructional contexts [67, 136], as described in Chapter 4.

Each group included one student in each of the following roles — the **Navigator**, who makes high-level decisions about the next implementation step based on group discussion; the **Driver**, who translates those decisions into code; the **Researcher**, who consults provided resources and external materials to assist the team in both reasoning and implementation; and the **Project Manager**, who ensures that all members follow and perform their assigned roles effectively. Roles rotate after each task to ensure that each student serves as the Driver exactly once during the project.

Prior to the experimental manipulation, students engaged in preparatory steps. They first reviewed explanatory text and video materials describing the collaborative programming environment, the role scaffold, and the overall structure of the project. Following this, they participated in a pilot project, conducted in the same environment and structured similarly to the main study. The pilot focused on lighter content to help students become familiar with the platform and collaborative structure. Groups were formed based on time availability, gathered via a pre-study survey. No two students from the same pilot group were placed together for the experimental manipulation. The positioning of the experimental study within the overall course is shown in Figure 8.1.

### 8.1.1 Data Collection

The collaborative project was immediately preceded and followed by a pre-test and post-test, respectively. Each test included two questions per learning objective (LO), totaling eight questions per test. Improvement in scores from pre- to post-test served as a measure of conceptual learning from the project. The individual programming project, due one week later, served as a measure of both conceptual and procedural knowledge, reflecting students' preparation for future problem-solving transfer. The positioning of these components is summarized in Figure 8.1.

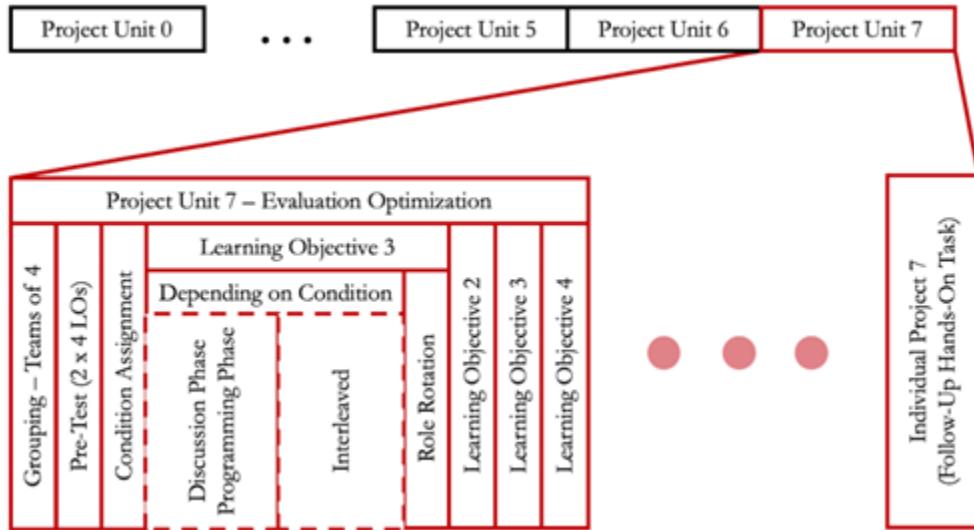


Figure 8.1: Course structure and positioning of experimental study — alignment of pre-test, post-test, collaborative project, and individual programming project

A total of 76 students participated in the collaborative project and the subsequent individual programming project. Ten groups (36 students) were assigned to the phased condition, and eleven groups (42 students) to the interleaved condition. The group sizes were not evenly split due to a few no-shows or course withdrawals. In a small number of cases, students worked in groups of three, with one student assuming responsibility for both the Project Manager and Researcher roles, as instructed in the modified role scaffold.

### 8.1.2 Experimental Manipulation

The study was conducted using a between-subjects design where students, in groups of 4, were assigned to two different conditions — the phased design used in prior work, and the novel interleaved design. One task is used to illustrate the two conditions. Other tasks followed a similar structure but target different learning objectives. Time-on-task for each project was controlled. Only the distribution of attention to problem solving and reflection differed. Thus, the question was what strategy for allocating attention was the best use of instructional time for producing learning gains.

#### Phased Design — Example-Based Reflection Followed by Programming

Students first participated in the collaborative reflection phase. Here, they were presented with a worked example solution in the form of pseudocode, or an algorithmic outline. The conversational agent prompted comparisons to possible alternative implementations, leading to a collaborative reflection about the pros and cons of each approach. Once the reflection concluded and consensus was reached about the approach

that the group would use, they entered the programming phase where they were assisted in their implementation through hints provided by the conversational agent.

Condition	Phased Design	Interleaved Design
Pseudocodes Presented	1 Pseudocode Presented	2 Pseudocodes Presented
Start of Project	Start of Collaborative Reflection Phase	Start of Interleaving
Reflection Prompt 1	Led by the Project Manager, take turns to discuss why it's important to zero gradient on each iteration of training in this implementation.	What's the difference between the two pseudocodes?
Reflection Prompt 2	As a follow-up, what will happen if you do not do it? Can you think of an alternate implementation that doesn't require it?	Why is this difference important?
Phase Transition (If Required)	Start of Programming Phase	No separation of phases necessary
Implementation Hint	For a full training loop example, have the researcher look up <a href="https://pytorch.org/docs">https://pytorch.org/docs</a>	For a full training loop example, have the researcher look up <a href="https://pytorch.org/docs">https://pytorch.org/docs</a>

Table 8.1: Task Flow and Example Conversational Agent-Prompts Across Conditions

### Interleaved Design — Reflection “in the Flow” of Programming

In this condition, students were presented with two worked example pseudocode solutions. Rather than introducing the alternative implementations as discussion prompts, students were asked to compare and contrast the two presented solutions and choose one that they would implement. This eliminated the need for a separate discussion phase by embedding the conceptual learning in the decision-making process between the two presented solutions. Similar to the phased condition, implementation assistance was provided through hints. Table 8.1 illustrates the flow of one task for students in this condition.

## 8.2 Analysis and Results

We first verified that learners in both conditions acquired conceptual knowledge from participating in the projects, as measured by pre- to post-test performance. We built a repeated measures ANOVA model with Condition (Phased vs Interleaved), Time Point (pre- vs post-), and Learning Objective (one of four) as independent variables, and test score as the dependent variable. We included all pairwise and three-way interaction terms as well. The full ANOVA model was significant ( $p = 0.001$ ), as was the effect of Learning Objective  $F(3, 548) = 7.1$ ,  $p < 0.0001$  and Time Point  $F(1, 548) = 5.4$ ,  $p < 0.05$ . However, neither Condition nor any of the interaction terms were significant. Thus, there were significant gains between pre- and post-test that were general across both conditions and across all learning objectives, though some learning objectives were harder than others.

	LO1 — Model- Building Workflow	LO2 — CPU vs. GPU Operations	LO3 — Model Evaluation	LO4 — Applying a Pre-Trained Model
Phased/Pre-test	0.79 (0.28 SD)	0.82 (0.39 SD)	0.94 (0.20 SD)	0.91 (0.29 SD)
Phased/Post-test	0.88 (0.25 SD)	0.94 (0.25 SD)	0.94 (0.25 SD)	0.91 (0.30 SD)
Interleaved/Pre- test	0.76 (0.32 SD)	0.74 (0.45 SD)	0.95 (0.16 SD)	0.92 (0.27 SD)
Interleaved/Post- test	0.88 (0.22 SD)	0.84 (0.37 SD)	0.97 (0.11 SD)	0.97 (0.16 SD)

Table 8.2: Raw test scores on conceptual knowledge for both conditions at pre-test and post-test across all four learning objectives (Scores out of 1.0; SD in parentheses)

Next, we tested for differential amounts of gain between conditions. We accomplished this using an ANCOVA model with Condition and Learning Objective as independent variables, pre-test score and time-on-task as covariates, and post-test score as the dependent variable. In this case, the correlation between pre- and post-test score was significant across all learning objectives ( $p < 0.0001$ ), but there was no significant difference in conceptual learning gain between the two conditions. Thus, the interleaved design did not show any signs of being detrimental for conceptual learning. In fact, the trend, though not significant, was for more learning to occur in the interleaved design.

Finally, we tested for differential performance on the subsequent individual programming project, which served as a hands-on transfer task. This project was graded across 9 criteria, ordered by difficulty. We tested the difference across criteria between conditions using an ANOVA model with Condition and Criterion as independent vari-

ables and score as the dependent variable. We also included the interaction term. There was a significant effect of Condition  $F(1, 666) = 13.0, p < 0.0005$ , Criterion  $F(8, 666) = 2.4, p < 0.05$ , and the Condition  $\times$  Criterion interaction  $F(8, 666) = 2.3, p < 0.05$ . The significant interaction effect indicates that the performance difference between conditions varied across criteria. Post-hoc analyses revealed that the advantage for the Interleaved condition was most pronounced and statistically significant on the final two, most difficult criteria, suggesting the interleaved approach particularly benefited performance on more challenging transfer tasks.

## 8.3 Discussion

The results of the study suggest that interleaving collaborative reflection “into the flow” of programming offers advantages over a phased approach, particularly for preparing students for future problem-solving transfer. While both conditions led to comparable conceptual learning gains, the interleaved design resulted in significantly better performance on a subsequent individual programming project, especially on more difficult components. An example conversation, shown below, illustrates how reflection occurred within the context of action in the interleaved condition.

- **Transactive Exchange**

*Student 1: “... I used ‘f.readlines()’ to read the wiki log file. It worked well on my own computer, but it caused a ‘MemoryError’ when I tested it on AWS ...”*

*Student 2: “The file object itself is an iterator. So if you ‘for x in file’, you get lines as x. This is a more Pythonic way than using ‘readline()’.”*

- **Non-Transactive Exchange**

*Student 1: “... I approached the problem by breaking it out into different modules and functions which made it possible to test different cases really fast. ...”*

*Student 2: “Well done!”*

*(Example from Interleaved Condition: In response to compare-and-contrast prompt from the conversational agent)*

*Navigator: “ok well the model has no predict function” (sic)*

*Researcher: “I’ll see what the primer says about predictions.”*

*Researcher: “You can do something along these lines: <ExampleCode>”*

*Driver, implements.*

*Navigator: “it says a list of the two logit scores returned by the prediction output, I don’t really know what that means” (sic)*

*Project Manager: “can we just try returning logits?”*

*Driver, implements, producing a specific error: “‘RobertaTokenizer’ object has no attribute ‘transform’”*

*Group, realizes the solution.*

*Researcher: “We just call it.”*

*Project Manager: "Ohh right."*

...

*OPE\_Bot: "You have passed this test case!"*

This exchange highlights the fluid movement between implementation attempts (e.g., the Driver's error message) and reflection prompted by considering the alternatives (e.g., the Navigator and Project Manager's dialogue about the prediction function and logits), characteristic of the interleaved design where reflection occurs directly in the context of problem-solving action.

Like the mechanism for conceptual learning through self-explanations, the reflection prompts required group members to explicitly articulate their mental models. The collaborative context then meant that misconceptions, when revealed, were addressed through discussion. Both conditions produced these discussions that resulted in conceptual learning, albeit in different forms — by reflecting over contrasting solutions in the interleaved design, or by responding to collaborative reflection prompts in the phased design. The interleaved design, however, increased exposure to reflection opportunities by not relegating access to the solutions only to the collaborative reflection phase. The reflection also occurred "in the context" of the solution, resulting in beneficial effects similar to those observed from self-explanations prompted during problem solving [7].

## 8.4 Chapter Discussion

At the end of the several studies presented in this and previous chapters, we have arrived at a design that maximizes exposure to problem solving as well as reflection opportunities from a project. Further, since the reflection happened in the context of programming and vice versa, there appeared to be a positive effect over and above that observed from separately tackling the two. The interleaved design, by virtue of allowing for the creation of a product in the process of learning, shows promise for satisfying the need for learning "in the flow," carrying important implications not only for computer science education in the higher-ed context but for the workplace learning context as well. The workplace learning context necessitates that learning happen "in the flow" of work and that distinct learning opportunities be provided only when absolutely necessary. This is discussed in the implications and future work presented in Chapter 9.

# Chapter 9

## Discussion, Contributions, and Future Work

### 9.1 Discussion and Implications

This dissertation investigated the central question of how worked example study compares with problem-solving practice for conceptual learning and individual project performance within the context of advanced computer science courses. The findings present significant implications for both the theory of example-based learning and the prevailing pedagogy in this domain.

A key finding confirms the predictions of example-based learning theory — spending more time studying worked examples leads to superior conceptual learning outcomes compared to engaging primarily in problem-solving practice. This holds true because, despite their advanced standing, learners in these courses are effectively novices concerning the specific, complex conceptual knowledge introduced in each new unit. Their prior expertise does not automatically transfer to these new concepts, making the guided, explanatory nature of worked examples more beneficial for initial understanding [78, 127].

However, the role of prior procedural knowledge — specifically, familiarity with the programming language used in the project — introduces important nuance. Programming proficiency involves both syntactic (procedural) knowledge and semantic (conceptual) understanding. Learners already possessing procedural fluency know the syntax and programming language concepts well. For them, while worked examples remain crucial for grasping the new domain-relevant concepts, a limited amount of problem-solving practice proves beneficial. This practice likely helps them bridge the gap between their newfound conceptual understanding and its practical application, allowing them to translate abstract concepts into concrete code effectively [146].

Conversely, learners without prior procedural knowledge in the specific language face the dual challenge of acquiring both syntax and semantics, in addition to domain-relevant conceptual knowledge. Typically, mastering all three within the confines of a single project would be exceedingly difficult. Yet, the studies indicate that these

advanced students leverage their existing knowledge of fundamental programming paradigms, enabling them to transfer concepts and learn the syntax of a new language relatively quickly. Even for this group, maximizing time with worked examples remains optimal for conceptual learning. While they require slightly more problem-solving practice than their procedurally knowledgeable peers to solidify both syntax and semantics, spending excessive time on problem-solving (more than half the allocated time) becomes sub-optimal [147, 149].

The observed order effects — where presenting worked examples first benefits conceptual learning, while problem-solving first aids individual project performance — can be interpreted through the lens of transfer-appropriate processing [110, 117]. This theory posits that learning is enhanced when the cognitive processes engaged during learning match those required during assessment. Studying worked examples first likely primes conceptual understanding, aligning well with conceptual tests. Conversely, starting with problem-solving may better prepare students for the procedural demands of the individual implementation project. Literature on example-based learning also suggests that a problem-solving-first approach might be viable when tasks exhibit low element interactivity [27, 28]. The project design, breaking down the overall goal into distinct tasks aligned with specific learning objectives, might have contributed to achieving lower element interactivity, potentially explaining the observed benefit of problem-solving first for project performance, especially when tasks closely mirrored those in the subsequent individual project.

Collectively, these findings challenge the deep-rooted reliance on minimally-guided problem-solving practice prevalent in advanced computer science education. The evidence strongly supports instructional designs prioritizing worked example study and reflection. However, the results also highlight that a one-size-fits-all approach is insufficient. Optimal design requires tailoring based on factors like prior procedural knowledge and the specific learning outcome prioritized (conceptual understanding vs. implementation skill). This dissertation demonstrates that such tailoring can be effectively achieved through two distinct avenues — adaptive AI-driven interventions and carefully crafted learning designs.

The innovative “interleaved” project design presented in Chapter 8 exemplifies the latter approach. By presenting strategically different worked examples and requiring students to compare them before translating pseudocode to code, this design cleverly integrates the cognitive benefits of both worked example reflection and problem-solving practice into a unified activity. It maximizes exposure to both reflection and application opportunities within a fixed time frame. Furthermore, because the reflection happens in the context of programming and vice versa, there appears to be a positive synergistic effect beyond tackling the two separately. This interleaved design, by virtue of allowing for the creation of a product (code) in the process of learning, shows particular promise for satisfying the need for learning “in the flow.” This carries important implications not only for computer science education in higher-ed but especially for the workplace learning context. Learners in the workplace adopt even more of an efficiency mindset, and learning opportunities must be engaging as they are largely optional. The interleaved design can potentially address these concerns by embedding learning into

productive work. With the advancement of LLMs specifically for programming, tasks such as onboarding to a new project, learning new implementation syntax, or understanding codebase contributions can be transformed into learning opportunities “in the flow” of work. LLMs could be prompted with context, like documentation and existing code, to craft worked examples (perhaps in pseudocode) for tasks assigned to software engineers, yielding both implementation progress and targeted learning. Early experiments exploring LLMs for codebase understanding and guided tours show promise in this direction [4, 14, 17], suggesting similar processes could create embedded learning opportunities.

Alternately, the Reinforcement Learning methodology developed and tested in Chapter 7 demonstrates a practical approach for data-driven adaptation *in vivo*. It allows instructional designers and researchers to move beyond static designs towards adaptive systems that optimize support (like prompt timing) based on observed effectiveness within the deployment context itself. This provides a mechanism to tailor interventions dynamically, implicitly accounting for factors like varying needs for problem-solving time due to prior knowledge. Future iterations could leverage LLMs to enhance this process; instead of relying solely on supervised classifiers of conversational indicators for episodic reward updates, LLMs might enable real-time evaluation of discussion quality or even direct adaptation of conversational agent behavior in response to the success or failure of an intervention. Of course, such real-time adaptation would need careful tuning to balance interrupting collaboration versus seeding transactive exchange opportunities. Furthermore, while early attempts at using LLMs for tutorial dialogue show promise, challenges like hallucination and the need for pedagogically sound training data remain [116]. Addressing these requires high-quality interaction data. In this regard, the several semesters of multi-party pedagogical conversation data collected throughout this dissertation work represent a valuable resource, especially as traditional data sources risk contamination from LLM outputs, potentially leading to model collapse [157]. This dataset could potentially support future work in training or evaluating educational AI systems, including automated analysis techniques like those using multi-agent LLM systems for thematic analysis [152].

## 9.2 Contributions

The research presented in this dissertation makes several significant contributions, spanning educational technology, learning science, and artificial intelligence in education, all framed within the iterative process of “Educational Experimentation At-Scale” introduced in Chapter 1 and depicted in Figure 1.1.

The foundational contribution is the development and integration of a robust platform for conducting educational experiments within authentic advanced computer science courses. Recognizing the inherent difficulties in collecting high-quality educational data at scale, we integrated the Bazaar AI conversational agent framework [2] into a novel team programming project interface, deployed via the Sail() platform (<https://sailplatform.org/>). This technological infrastructure addresses multiple challenges

- 
- It provides a means to instrument learning environments for collecting fine-grained process data (e.g., code contributions, chat interactions, reflections) alongside outcome measures (pre/post-tests, project performance), moving beyond infrequent, coarse-grained assessments. This aligns with the need to *Design for Measurement*. The ability to collect rich data like student reflections also enables downstream analysis and tool development, such as dashboards to help instructors navigate reflections at scale [12].
  - It enables the rapid iteration and deployment of different instructional interventions (facilitated by AI agents) across multiple courses and semesters, crucial for the *Iterative Design Process* inherent in design-based research and the “at-scale” approach.
  - Its modular and extensible design has proven durable, allowing subsequent research to build upon it by incorporating multimodal interactions [191], portable interfaces [185], and advanced AI support using large language models [111].

This platform directly facilitated the progression through the experimentation cycle — from understanding the context and designing interventions to running experiments and refining hypotheses.

A second major contribution lies in the empirical findings and their direct pedagogical implications. Initial data collected using the instrumented platform revealed the ineffectiveness of existing, traditional project designs in fostering measurable conceptual learning gains. This crucial baseline finding underscored the need for change and justified the subsequent effort invested in a learning science-informed redesign. The process of redesign itself, detailed in Chapter 5 and guided by the KLI framework [88], highlighted the importance of aligning activities with *Granular Learning Objectives* and incorporating principles like *Scaffolded Collaboration* and structured *Reflection*. The subsequent comparative experiments, presented in Chapter 6, provided strong evidence favoring worked example study over extensive problem-solving practice for conceptual learning in this advanced context. This directly challenges long-standing pedagogical assumptions and practices in computer science education, which often prioritize unguided “learning by doing” under the misconception that worked examples constitute passive learning [9]. These findings offer actionable *Lessons for Practitioners*, demonstrating that evidence-based redesign, though potentially effort-intensive initially, leads to more effective learning outcomes. The positive impact on student learning experiences across several cohorts at Carnegie Mellon University provides concrete evidence of this contribution’s real-world value. Furthermore, the research highlights the critical need to *Balance Learning and Productivity Goals* in project-based settings.

Finally, this dissertation contributes novel strategies for tailoring instruction in response to empirical findings, advancing both AI in Education and Instructional Design.

- For the AIED community, facing the “small data” challenge common in educational RL contexts and the ethical constraints surrounding exploration, we developed and validated solutions rooted in deep contextual understanding, as shown in Chapter 7. The *Informed State Design*, constrained by learning science princi-

bles and domain knowledge, mitigates risks associated with detrimental learning states. The *Episodic Parameter Update* strategy offers a method to learn effective policies from limited interaction data, viable due to the high-quality data generated by the well-designed project [148]. This work demonstrates a practical methodology for leveraging RL principles for in vivo tuning. While the algorithms used were simpler than those often employed in simulation-based educational RL, they proved effective when combined with pedagogical knowledge, careful problem formulation, and realistic constraints, underscoring the importance of context-aware AI design in education. This RL-based approach specifically enables tailoring by learning optimal durations or timings of interventions based on observed interaction patterns, implicitly capturing proxies for group or individual states like prior knowledge levels.

- From the learning sciences and instructional design community, we proposed and validated the innovative “interleaved” project design (Chapter 8). This design offers a practical, non-AI-based solution for integrating reflection and practice, addressing the nuanced needs identified in the experimental findings by maximizing exposure to both within a unified activity structure.

These strategies exemplify how understanding the underlying cognitive processes driving learning outcomes can lead to effective and novel interventions. They demonstrate the power of combining data-driven insights with theory-informed design to achieve measurable impact.

### 9.3 Future Work

The findings and methodologies presented in this dissertation open several promising avenues for future research, building upon the established theoretical and technological groundwork.

One significant direction involves leveraging recent advances in large language models (LLMs). While this dissertation focused on more structured AI approaches, LLMs offer potential for more sophisticated personalization and adaptation. As subsequent work has explored, LLMs can be used for personalized feedback, solution-path specific prompting [111, 151], or even generating contrasting cases to support worked example reflection [112]. However, as observed in some of this follow-up work, simply adding more sophisticated reflection support does not guarantee better learning; the time spent on reflection can negatively impact task completion without necessarily improving learning outcomes if not carefully balanced. This reinforces a core finding from this dissertation — theoretically grounded hypotheses about the optimal distribution of time and cognitive effort between activities (like example study vs. problem-solving) are paramount and often have more impact than the specifics of the technology delivering the intervention. Nonetheless, LLMs can significantly accelerate the development of theory-informed interventions, thereby speeding up the cycle of empirical testing and theoretical refinement within naturalistic environments, like the platform developed

here.

A key area where LLMs might enhance the interventions studied here is in achieving more dynamic adaptation. Theoretically, LLMs could be employed to more effectively “probe the exact status of the learners” by analyzing their natural language explanations, justifications, or chat contributions in richer ways than current methods. This deeper, real-time understanding could enable highly adaptive fading strategies — dynamically transitioning from worked examples to problem-solving tasks precisely when readiness is detected. Such capabilities might allow for more accurate identification of expertise-reversal onset [134], triggering personalized interventions. The principle of fading support as expertise grows is well-established; LLMs offer a potentially powerful tool for implementing this principle more effectively based on real-time evidence. Automated analysis of the rich conversational data collected by our platform, perhaps using emerging multi-agent LLM systems for thematic analysis [152], could also help identify patterns indicative of learner status or group dynamics to inform such adaptations.

Another avenue is to explore tailoring based on learner characteristics beyond the prior procedural knowledge investigated here. While prior knowledge proved critical, other aspects like self-efficacy, cognitive load tolerance, learning goals, motivation, or metacognitive skills might interact significantly with different instructional strategies. Investigating methods for assessing these characteristics (potentially through surveys, behavioral traces, or even LLM analysis of student interactions) and adapting support accordingly is an important direction. LLM-based approaches are already beginning to address variations in learning goals, for instance, through “socratic dialogue” modes designed to help learners articulate and pursue their specific interests [10].

While this dissertation focused on learning outcomes within the academic context, investigating the translation to professional practice in the workplace remains a crucial, albeit challenging, next step. Addressing the inherent attribution problem requires creative methodologies. However, the approach used in our RL experiments — using theoretically grounded proximal indicators (conversational patterns) as proxies for distal rewards (learning) — offers a parallel. By leveraging access to both fine-grained process data from educational interventions and, where feasible and ethically appropriate, longer-term workplace outcome data (e.g., performance, retention), correlational analyses could identify potential links. These correlations, while not causal, could form valuable hypotheses about the lasting impact of specific educational designs or interaction patterns, guiding future, more targeted studies.

Finally, the iterative cycle of “Educational Experimentation At-Scale” (Figure 1.1) should continue, building on the RL-based methodology developed here. The approach proved effective for tuning a single parameter based on proximal rewards. Future work could explore optimizing multiple parameters simultaneously, requiring larger cohorts or more data-efficient algorithms. More complex state representations incorporating real-time group dynamics could enable more context-sensitive adaptations.

The most provocative implication of the work in this dissertation and follow-up work since, is the need for simulated environments that mimic student interactions and allow for rapid, large scale experiments and data collection. Notwithstanding

the caveats around model collapse [157] mentioned above, simulated interactions have been tested with mixed results [1, 200, 202]. The question, as always, is whether simulated agents can compare to the fidelity of real humans. Further, if simulated agents are prompted to behave based on existing theoretical understanding about how humans learn, then the question of whether they can produce novel interaction behaviors to further theoretical understanding is an open question. Emergent behavior from LLMs has certainly been reported but debate exists about whether such behavior is truly emergent, and, in our case, whether it is reflective of how humans learn. Ultimately, for now, the most reliable path forward likely involves continued experimentation in human environments, perhaps accelerated by using LLMs to more quickly translate theoretical insights into practical interventions for testing. Validating the core findings and methods across different domains and populations remains essential for establishing generalizability.

## 9.4 Conclusion

This dissertation embarked on an investigation into the effectiveness of worked examples versus problem-solving practice in advanced computer science education, challenging prevailing pedagogical norms. Through a rigorous, multi-stage process encompassing iterative design, technological integration, and large-scale experimentation within authentic learning environments, we demonstrated that prioritizing worked example study yields significant benefits for conceptual learning. The findings also highlighted important nuances related to prior procedural knowledge, task ordering, and the potential benefits of interleaved designs, indicating that optimal instructional strategies require careful tailoring to the specific learners and learning goals.

The work provides a robust technological framework (the instrumented Sail() platform with the Bazaar agent) for continued educational experimentation at scale, enabling the collection of rich data necessary for both analysis and adaptation, as demonstrated by related work on instructor dashboards [12]. It yields actionable, evidence-based pedagogical recommendations for computer science educators, particularly regarding the value of worked examples and structured reflection. Furthermore, it contributes two distinct strategies for achieving tailored learning support — the practical and effective non-AI “interleaved” instructional design, and a novel, viable methodology for using Reinforcement Learning principles to perform *in vivo* optimization of intervention parameters under realistic educational constraints. This RL-based approach, potentially enhanced by LLMs for reward evaluation or automated analysis [152], offers a promising pathway for creating more adaptive and effective learning systems by directly leveraging data from the deployment context.

By grounding AI development in learning science theory and contextual understanding, and by using empirical data to refine both theory and practice, this research exemplifies a powerful synergy. It demonstrates how technology-enhanced learning environments can serve not only as delivery platforms but also as sophisticated instruments for scientific inquiry and continuous improvement. This dissertation lays a solid

foundation — theoretically, methodologically, and technologically — for future work aiming to further enhance learning in complex domains, bridge the gap between educational research and practice, and ultimately contribute to scalable, real-world impact on student success, both in academic settings and potentially in the workplace.

# Bibliography

- [1] Zahra Abbasiantaeb, Yifei Yuan, Evangelos Kanoulas, and Mohammad Aliannejadi. Let the llms talk: Simulating human-to-human conversational qa via zero-shot llm-to-llm interactions. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pages 8–17, 2024. 9.3
- [2] David Adamson and Carolyn Penstein Rosé. Coordinating multi-dimensional support in collaborative conversational agents. In *Intelligent Tutoring Systems: 11th International Conference, ITS 2012, Chania, Crete, Greece, June 14–18, 2012. Proceedings* 11, pages 346–351. Springer, 2012. 1.1, 4.2.1, 9.2
- [3] David Adamson, Gregory Dyke, Hyeju Jang, and Carolyn Penstein Rosé. Towards an agile approach to adapting dynamic collaboration support to student needs. *International Journal of Artificial Intelligence in Education*, 24(1):92–124, 2014. 2.2.2, 4.1, 4.2, 4.2.1, 8.1
- [4] Elijah Kayode Adejumo and Brittany Johnson. Towards leveraging llms for reducing open source onboarding information overload. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pages 2210–2214, 2024. 9.1
- [5] Ravindra K Ahuja and James B Orlin. A fast and simple algorithm for the maximum flow problem. *Operations Research*, 37(5):748–759, 1989. 5.3.1
- [6] Hua Ai, Rohit Kumar, Dong Nguyen, Amrut Nagasunder, and Carolyn P Rosé. Exploring the effectiveness of social capabilities and goal alignment in computer supported collaborative learning. In *International Conference on Intelligent Tutoring Systems*, pages 134–143. Springer, 2010. 5.3.1
- [7] Vincent AWMM Aleven and Kenneth R Koedinger. An effective metacognitive strategy: Learning by doing and explaining with a computer-based cognitive tutor. *Cognitive science*, 26(2):147–179, 2002. 8, 8.3
- [8] Michael Allen. The successive approximation model (sam). In *Trends and Issues in Instructional Design and Technology*, pages 67–81. Routledge, 2012. 1, 3.2
- [9] Anonymous. Ask hn: How to develop a first-principles understanding of cs, 2025. URL <https://news.ycombinator.com/item?id=34943837>. Accessed: April 14, 2025. 1, 9.2
- [10] Anthropic. Introducing claude for education, Apr 2025. URL <https://www.anthropic.com/news/introducing-claude-for-education>. Accessed: April 20,

2025. 9.3

- [11] E Aronson. The jigsaw classroom, 1978. 2.2.1
- [12] Yuya ASANO, Sreecharan SANKARANARAYANAN, Christopher BOGART, et al. A thematic summarization dashboard for navigating student reflections at scale. In *International Conference on Computers in Education*, 2021. 9.2, 9.4
- [13] Markel Sanz Ausin. Leveraging deep reinforcement learning for pedagogical policy induction in an intelligent tutoring system. In *In: Proceedings of the 12th International Conference on Educational Data Mining (EDM 2019)*,, 2019. 7.2, 7.5.1
- [14] Maider Azanza, Juanan Pereira, Arantza Irastorza, and Aritz Galdos. Can llms facilitate onboarding software developers? an ongoing industrial case study. In *2024 36th International Conference on Software Engineering Education and Training (CSEE&T)*, pages 1–6. IEEE, 2024. 9.1
- [15] Prashant Baheti, Edward Gehringer, and David Stotts. Exploring the efficacy of distributed pair programming. In Don Wells and Laurie Williams, editors, *Extreme Programming and Agile Methods — XP/Agile Universe 2002*, pages 208–220, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-45672-8. 4.1.1
- [16] Ryan Shaun Baker, Albert T Corbett, Kenneth R Koedinger, and Angela Z Wagner. Off-task behavior in the cognitive tutor classroom: when students game the system. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 383–390. Acm, 2004. 5.1.3
- [17] Martin Balfroid, Benoît Vanderose, and Xavier Devroey. Towards llm-generated code tours for onboarding. In *Proceedings of the Third ACM/IEEE International Workshop on NL-based Software Engineering*, pages 65–68, 2024. 9.1
- [18] Brigid Barron. Achieving coordination in collaborative problem-solving groups. *The journal of the learning sciences*, 9(4):403–436, 2000. 2.2.1
- [19] Brigid Barron. When smart groups fail. *The journal of the learning sciences*, 12(3): 307–359, 2003. 2.2
- [20] Joseph E. Beck and Yue Gong. Wheel-spinning: Students who fail to master a skill. In H. Chad Lane, Kalina Yacef, Jack Mostow, and Philip Pavlik, editors, *Artificial Intelligence in Education*, pages 431–440, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-39112-5. 5.1.3
- [21] Nancy Bell, Michèle Grossen, and Anne-Nelly Perret-Clermont. *Sociocognitive conflict and intellectual growth*. Jossey-Bass, 1985. 2.2.1, 5.4.1
- [22] Marvin W Berkowitz and John C Gibbs. Measuring the developmental features of moral discussion. *Merrill-Palmer Quarterly* (1982-), pages 399–410, 1983. 5.3
- [23] Jennifer A Bivens and Laura E Berk. A longitudinal study of the development of elementary school children’s private speech. *Merrill-Palmer Quarterly* (1982-), pages 443–463, 1990. 2.2
- [24] Robert Maribe Branch and İlhan Varank. *Instructional design: The ADDIE approach*, volume 722. Springer, 2009. 1, 3.2

- [25] Jim Buchan and Mark Pearl. Leveraging the mob mentality: An experience report on mob programming. In *Proceedings of the 22Nd International Conference on Evaluation and Assessment in Software Engineering 2018, Ease'18*, pages 199–204, New York, NY, USA, 2018. Acm. ISBN 978-1-4503-6403-4. doi: 10.1145/3210459.3210482. URL <http://doi.acm.org/10.1145/3210459.3210482>. 4.1.1
- [26] Sharon M Carver. Cognition and instruction: Enriching the laboratory school experience of children, teachers, parents, and undergraduates. In *Cognition and instruction: Twenty-five years of progress*, pages 385–426. Lawrence Erlbaum Associates, 2001. 2.2.2, 3.2
- [27] Ouhao Chen, Endah Retnowati, and Slava Kalyuga. Element interactivity as a factor influencing the effectiveness of worked example–problem solving and problem solving–worked example sequences. *British Journal of Educational Psychology*, 90:210–223, 2020. 2.1.2, 9.1
- [28] Ouhao Chen, Fred Paas, and John Sweller. A cognitive load theory approach to defining and measuring task complexity through element interactivity. *Educational Psychology Review*, 35(2):63, 2023. 2.1.2, 9.1
- [29] Michelene TH Chi, Paul J Feltovich, and Robert Glaser. Categorization and representation of physics problems by experts and novices. *Cognitive science*, 5(2):121–152, 1981. 2.1
- [30] Min Chi, Pamela Jordan, Kurth VanLehn, and Moses Hall. Reinforcement learningbased feature selection for developing pedagogically effective tutorial dialogue tactics. In *Educational Data Mining 2008*. Citeseer, 2008. 7.2
- [31] Min Chi, Kurt VanLehn, and Diane Litman. Do micro-level tutorial decisions matter: Applying reinforcement learning to induce pedagogical tutorial tactics. In *International conference on intelligent tutoring systems*, pages 224–234. Springer, 2010. 7.2
- [32] Min Chi, Kurt VanLehn, Diane Litman, and Pamela Jordan. Empirically evaluating the application of reinforcement learning to the induction of effective and adaptive pedagogical strategies. *User Modeling and User-Adapted Interaction*, 21(1):137–180, 2011. 7.2
- [33] Derrick Coetzee, Armando Fox, Marti A Hearst, and Bjoern Hartmann. Chatrooms in moocs: all talk and no action. In *Proceedings of the first ACM conference on Learning scale conference*, pages 127–136. Acm, 2014. 2.2.1
- [34] Elisabeth G Cohen and Rachel A Lotan. *Designing Groupwork: Strategies for the Heterogeneous Classroom Third Edition*. Teachers College Press, 2014. 2.2.1
- [35] Semra Comu, Josh Iorio, John E Taylor, and Carrie Sturts Dossick. Quantifying the impact of facilitation on transactive memory system formation in global virtual project networks. *Journal of construction Engineering and Management*, 139(3):294–303, 2013. 5.3
- [36] Cristina Conati and Kurt VanLehn. Teaching meta-cognitive skills: Implementa-

tion and evaluation of a tutoring system to guide self-explanation while learning from examples. In *Artificial intelligence in education*, pages 297–304. IOS Press, 1999. 8

- [37] Graham Cooper and John Sweller. Effects of schema acquisition and rule automation on mathematical problem-solving transfer. *Journal of educational psychology*, 79(4):347, 1987. 2.1
- [38] Catherine H Crouch and Eric Mazur. Peer instruction: Ten years of experience and results. *American journal of physics*, 69(9):970–977, 2001. 2.2.1
- [39] Sarah D’Angelo and Andrew Begel. Improving communication between pair programmers using shared gaze awareness. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, Chi ’17, pages 6245–6290, New York, NY, USA, 2017. Acm. ISBN 978-1-4503-4655-9. doi: 10.1145/3025453.3025573. URL <http://doi.acm.org/10.1145/3025453.3025573>. 4.1.1
- [40] Stephen Davies, Jennifer A Polack-Wahl, and Karen Anewalt. A snapshot of current practices in teaching the introductory programming sequence. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 625–630, 2011. 2.1
- [41] Ton De Jong and Wouter R Van Joolingen. Scientific discovery learning with computer simulations of conceptual domains. *Review of educational research*, 68(2):179–201, 1998. 2.2.1
- [42] Pierre Dillenbourg. Computerized scripts for enhancing collaborative learning. In *International Conference on Open and Online Learning ICOOL 2003*, pages 113–133, 2003. 2.2.1
- [43] Pierre Dillenbourg and Fabrice Hong. The mechanics of cscl macro scripts. *International Journal of Computer-Supported Collaborative Learning*, 3(1):5–23, 2008. 2.2.2
- [44] Pierre Dillenbourg and Patrick Jermann. Technology for classroom orchestration. *New science of learning: Cognition, computers and collaboration in education*, pages 525–552, 2010. 2.2.1
- [45] Willem Doise, Gabriel Mugny, and Anne-Nelly Perret-Clermont. Social interaction and the development of cognitive operations. *European journal of social psychology*, 5(3):367–383, 1975. 2.2.1, 5.4.1
- [46] Shayan Doroudi, Vincent Aleven, and Emma Brunskill. Robust evaluation matrix: Towards a more principled offline exploration of instructional policies. In *Proceedings of the fourth (2017) ACM conference on learning scale*, pages 3–12, 2017. 7.2
- [47] Shayan Doroudi, Vincent Aleven, and Emma Brunskill. Where’s the reward? *International Journal of Artificial Intelligence in Education*, 29(4):568–620, 2019. 7.2
- [48] Gregory Dyke, Iris Howley, David Adamson, Rohit Kumar, and Carolyn Penstein Rosé. Towards academically productive talk supported by conversational agents. In *Productive multivocality in the analysis of group interactions*, pages 459–

476. Springer, 2013. 4.1

- [49] Barbara J Ericson, Lauren E Margulieux, and Jochen Rick. Solving parsons problems versus fixing and writing code. In *Proceedings of the 17th koli calling international conference on computing education research*, pages 20–29, 2017. 8
- [50] Mohammadreza Farrokhnia, Héctor J Pijeira-Díaz, Omid Noroozi, and Javad Hatami. Computer-supported collaborative concept mapping: The effects of different instructional designs on conceptual understanding and knowledge co-construction. *Computers & Education*, 142:103640, 2019. 5.1.1
- [51] Mark Felton, Merce Garcia-Mila, Constanza Villarroel, and Sandra Gilabert. Arguing collaboratively: Argumentative discourse types and their potential for knowledge building. *British Journal of Educational Psychology*, 85(3):372–386, 2015. 5.1.4
- [52] Oliver Ferschke, Diyi Yang, Gaurav Tomar, and Carolyn Penstein Rosé. Positive impact of collaborative chat participation in an edx mooc. In *International Conference on Artificial Intelligence in Education*, pages 115–124. Springer, 2015. 2.2.1
- [53] Frank Fischer, Ingo Kollar, Karsten Stegmann, and Christof Wecker. Toward a script theory of guidance in computer-supported collaborative learning. *Educational psychologist*, 48(1):56–66, 2013. 2.2.2, 8
- [54] Frank Fischer, Ingo Kollar, Karsten Stegmann, Christof Wecker, and Jan Zottmann. Collaboration scripts in computer-supported collaborative learning. In *The International Handbook of Collaborative Learning*, pages 415–431. Routledge, 2013. 2.2.2
- [55] Ellice A Forman and Myra J Kraker. The social origins of logic: The contributions of piaget and vygotsky. *New Directions for Child and Adolescent Development*, 1985(29):23–39, 1985. 2.2
- [56] Qing-Ke Fu and Gwo-Jen Hwang. Trends in mobile technology-supported collaborative learning: A systematic review of journal publications from 2007 to 2016. *Computers & Education*, 119:129–143, 2018. 2.2.1
- [57] Arpita Ghosh and Jon Kleinberg. Incentivizing participation in online forums for education. In *Proceedings of the fourteenth ACM conference on Electronic commerce*, pages 525–542. Acm, 2013. 5.1.1
- [58] Michel Gilly. 10-mécanismes psychosociaux des constructions cognitives: perspectives de recherche à l'âge scolaire. In *Développement et fonctionnement cognitifs chez l'enfant*, pages 201–222. Presses Universitaires de France, 1990. 2.2, 5.4.1
- [59] Susan L Golbeck and R DeLisi. Implications of piagetian theory for peer learning. *Cognitive perspectives on peer learning*, pages 3–37, 1999. 2.2.1, 5.4.1
- [60] Gahgene Gweon, Carolyn Rose, Regan Carey, and Zachary Zaiss. Providing support for adaptive scripting in an on-line collaborative learning environment. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 251–260, 2006. 2.2.3

- [61] Gahgene Gweon, S Jeon, J Lee, and Carolyn P Rosé. Diagnosing problems in student project groups. *Analyzing collaborative interactions in CSCL: Methods, approaches and issues*, pages 293–318, 2011. 4
- [62] Gahgene Gweon, Mahaveer Jain, John McDonough, Bhiksha Raj, and Carolyn P Rosé. Measuring prevalence of other-oriented transactive contributions using an automated measure of speech style accommodation. *International Journal of Computer-Supported Collaborative Learning*, 8:245–265, 2013. 5.3.1
- [63] Gahgene Gweon, Soojin Jun, Susan Finger, and Carolyn Penstein Rosé. Towards effective group work assessment: even what you don't see can bias you. *International Journal of Technology and Design Education*, 27:165–180, 2017. 4
- [64] Regine Hampel and Mirjam Hauck. Towards an effective use of audio conferencing in distance language courses. *Language Learning and Technology*, 2004. 4.2
- [65] Rachel Harsley, Davide Fossati, Barbara Di Eugenio, and Nick Green. Interactions of individual and pair programmers with an intelligent tutoring system for computer science. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, Sigmse '17, pages 285–290, New York, NY, USA, 2017. Acm. ISBN 978-1-4503-4698-6. doi: 10.1145/3017680.3017786. URL <http://doi.acm.org/10.1145/3017680.3017786>. 4.1.1
- [66] Khe Foon Hew and Wing Sum Cheung. Audio-based versus text-based asynchronous online discussion: Two case studies. *Instructional Science*, 41(2):365–380, 2013. 4.2
- [67] Michael Hilton and Sreecharan Sankaranarayanan. Online mob programming: Effective collaborative project-based learning. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, Sigmse '19, page 1283, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450358903. doi: 10.1145/3287324.3293774. URL <https://doi.org/10.1145/3287324.3293774>. 5.1, 8.1
- [68] Cindy E Hmelo, Mark Guzdial, and Jennifer Turns. Computer-support for collaborative learning: Learning to support student engagement. *Journal of Interactive Learning Research*, 9(2):107, 1998. 2.2.4
- [69] Cindy E Hmelo-Silver. Analyzing collaborative knowledge construction: Multiple methods for integrated understanding. *Computers & Education*, 41(4):397–420, 2003. 2.2
- [70] Loes Hogenkamp, Alieke M van Dijk, and Tessa HS Eysink. Analyzing socially shared regulation of learning during cooperative learning and the role of equal contribution: a grounded theory approach. *Education Sciences*, 11(9):512, 2021. 2.2
- [71] Jeroen Janssen and Paul A Kirschner. Applying collaborative cognitive load theory to computer-supported collaborative learning: Towards a research agenda. *Educational Technology Research and Development*, 68(2):783–805, 2020. 2.2
- [72] Jeroen Janssen, Femke Kirschner, Gijsbert Erkens, Paul A Kirschner, and Fred

- Paas. Making the black box of collaborative learning transparent: Combining process-oriented and cognitive load approaches. *Educational psychology review*, 22: 139–154, 2010. 2.2.1
- [73] Sanna Järvelä, Paivi Häkkinen, Maarit Arvaja, and Piritta Leinonen. Instructional support in cscl. In *What we know about CSCL*, pages 115–139. Springer, 2004. 2.2.1
- [74] David W Johnson and Roger T Johnson. Structuring groups for cooperative learning. *Organizational Behavior Teaching Review*, 9(4):8–17, 1984. 4.1.3
- [75] Mahesh Joshi and Carolyn Penstein Rosé. Using transactivity in conversation for summarization of educational dialogue. In *Workshop on Speech and Language Technology in Education*. Citeseer, 2007. 5.3.1
- [76] An Ju and Armando Fox. Teamscope: measuring software engineering processes with teamwork telemetry. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, pages 123–128. Acm, 2018. 4
- [77] Slava Kalyuga. The expertise reversal effect. In *Managing cognitive load in adaptive multimedia learning*, pages 58–80. IGI Global, 2009. 2.1
- [78] Slava Kalyuga, Paul Chandler, Juhani Tuovinen, and John Sweller. When problem solving is superior to studying worked examples. *Journal of educational psychology*, 93(3):579, 2001. 1, 2.1, 2.2.3, 9.1
- [79] Elaine Kant and Allen Newell. Problem solving techniques for the design of algorithms. *Information Processing & Management*, 20(1-2):97–118, 1984. 2.1.1
- [80] Herez Moise Kattan, Frederico Oliveira, Alfredo Goldman, and Joseph William Yoder. Mob programming: The state of the art and three case studies of open source software. In *Brazilian Workshop on Agile Methods*, pages 146–160. Springer, 2017. 4.1.1
- [81] Alison King. Ask to think-tel why: A model of transactive peer tutoring for scaffolding higher level complex learning. *Educational psychologist*, 32(4):221–235, 1997. 2.2, 2.2.3
- [82] Alison King. Structuring peer interaction to promote high-level cognitive processing. *Theory into practice*, 41(1):33–39, 2002. 2.2.1
- [83] Alison King. Scripting collaborative learning processes: A cognitive perspective. In *Scripting computer-supported collaborative learning: Cognitive, computational and educational perspectives*, pages 13–37. Springer, 2007. 2.2.1
- [84] Femke Kirschner, Fred Paas, and Paul A Kirschner. A cognitive load approach to collaborative learning: United brains for complex tasks. *Educational Psychology Review*, 21(1):31–42, 2009. 2.2.1
- [85] Paul A Kirschner, John Sweller, and Richard E Clark. Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational psychologist*, 41(2):75–86, 2006. 2.1

- [86] René F Kizilcec and Emily Schneider. Motivation as a lens to understand online learners: Toward data-driven design with the olei scale. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 22(2):6, 2015. 2.2.1, 4.2
- [87] Lars Kobbe, Armin Weinberger, Pierre Dillenbourg, Andreas Harrer, Raija Hämäläinen, Päivi Häkkinen, and Frank Fischer. Specifying computer-supported collaboration scripts. *International Journal of Computer-Supported Collaborative Learning*, 2(2):211–224, 2007. 2.2.1, 2.2.2
- [88] Kenneth R Koedinger, Albert T Corbett, and Charles Perfetti. The knowledge-learning-instruction framework: Bridging the science-practice chasm to enhance robust student learning. *Cognitive science*, 36(5):757–798, 2012. 1.1, 2.2.2, 3.2, 5.5.1, 9.2
- [89] Ingo Kollar, Frank Fischer, and Friedrich W Hesse. Collaboration scripts—a conceptual analysis. *Educational Psychology Review*, 18(2):159–185, 2006. 2.2.2
- [90] Ingo Kollar, Stefan Ufer, Elisabeth Reichersdorfer, Freydis Vogel, Frank Fischer, and Kristina Reiss. Effects of collaboration scripts and heuristic worked examples on the acquisition of mathematical argumentation skills of teacher students with different levels of prior achievement. *Learning and Instruction*, 32:22–36, 2014. 2.2.3
- [91] Yasmine Kotturi, Chinmay E Kulkarni, Michael S Bernstein, and Scott Klemmer. Structure and messaging techniques for online peer learning systems that increase stickiness. In *Proceedings of the Second (2015) ACM Conference on Learning Scale*, pages 31–38. Acm, 2015. 2.2.1
- [92] Koojana Kuladinithi, Leonard Fisser, Konrad Fuger, Daniel Stolpmann, Zeynep Vatandas, Andreas Timm-Giel, and Axel Dürkop. Online teaching of project-based learning courses: issues, challenges and outcomes. In *ACM SIGCOMM 2020*. ACM Special Interest Group on Data Communication (SIGCOMM), 2020. 2.2
- [93] Rohit Kumar, Carolyn Penstein Rosé, Yi-Chia Wang, Mahesh Joshi, and Allen Robinson. Tutorial dialogue as adaptive collaborative learning support. *Frontiers in artificial intelligence and applications*, 158:383, 2007. 4.1
- [94] Rohit Kumar, Hua Ai, Jack L Beuth, and Carolyn P Rosé. Socially capable conversational tutors can be effective in collaborative learning situations. In *International Conference on Intelligent Tutoring Systems*, pages 156–164. Springer, 2010. 4.1
- [95] Rachel Lam and Manu Kapur. Preparation for future collaboration: Cognitively preparing for learning from collaboration. *The Journal of Experimental Education*, 86(4):546–559, 2018. 5.1.1
- [96] Jimmie Leppink, Fred Paas, Tamara Van Gog, Cees PM van Der Vleuten, and Jeroen JG Van Merriënboer. Effects of pairs of problems and examples on task performance and different types of cognitive load. *Learning and instruction*, 30: 32–42, 2014. 2.1.2
- [97] Esther Levin, Roberto Pieraccini, and Wieland Eckert. Using markov decision

- process for learning dialogue strategies. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98* (Cat. No. 98CH36181), volume 1, pages 201–204. Ieee, 1998. 7.4
- [98] Xiongyi Liu, Brian Doore, and Lan Li. Scaffolding knowledge co-construction in web-based discussions through message labeling. In *Society for Information Technology & Teacher Education International Conference*, pages 3041–3046. Association for the Advancement of Computing in Education (AACE), 2008. 4.2
- [99] Ulrika Malmgren. Remote mob programming set up. <https://natoocktesting.wordpress.com/2017/01/30/remote-mob-programming-set-up/>, 2017. Accessed: 2018-08-30. 4.1.1
- [100] Lauren E Margulieux and Richard Catrambone. Improving problem solving with subgoal labels in expository text and worked examples. *Learning and Instruction*, 42:58–71, 2016. 1.1, 5.3.4
- [101] Lauren E Margulieux, Mark Guzdial, and Richard Catrambone. Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications. In *Proceedings of the ninth annual international conference on International computing education research*, pages 71–78, 2012. 1.1, 5.3.4
- [102] Lauren E Margulieux, Briana B Morrison, and Adrienne Decker. Design and pilot testing of subgoal labeled worked examples for five core concepts in cs1. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, pages 548–554, 2019. 2.1
- [103] Tanya J McGill and Simone E Volet. A conceptual framework for analyzing students' knowledge of programming. *Journal of research on Computing in Education*, 29(3):276–297, 1997. 2.1.1
- [104] Susan I McMahon. *Book Club: A case study of a group of fifth graders as they participate in a literature-based reading program.(Volumes I and II)*. Michigan State University, 1992. 2.2.1
- [105] Jay McTighe and Ronald S Thomas. Backward design for forward action. *Educational leadership*, 60(5):52–55, 2003. 1, 3.2
- [106] Stephan Mende, Antje Proske, and Susanne Narciss. Individual preparation for collaborative learning: Systematic review and synthesis. *Educational Psychologist*, 56(1):29–53, 2021. 5.1.1, 5.1.4
- [107] Muhsin Menekse and Michelene TH Chi. The role of collaborative interactions versus individual construction on students' learning of engineering concepts. *European Journal of Engineering Education*, 44(5):702–725, 2019. 2.2
- [108] Sarah Michaels, Catherine O'Connor, and Lauren B Resnick. Deliberative discourse idealized and realized: Accountable talk in the classroom and in civic life. *Studies in philosophy and education*, 27:283–297, 2008. 2.2.2
- [109] Mariel Miller and Allyson Hadwin. Scripting and awareness tools for regulating collaborative learning: Changing the landscape of support in cscl. *Computers in*

- [110] C Donald Morris, John D Bransford, and Jeffery J Franks. Levels of processing versus transfer appropriate processing. *Journal of verbal learning and verbal behavior*, 16(5):519–533, 1977. 9.1
- [111] Atharva Naik, Jessica Ruhan Yin, Anusha Kamath, Qianou Ma, Sherry Tong-shuang Wu, Charles Murray, Christopher Bogart, Majd Sakr, and Carolyn P Rose. Generating situated reflection triggers about alternative solution paths: A case study of generative ai for computer-supported collaborative learning. In *International Conference on Artificial Intelligence in Education*, pages 46–59. Springer, 2024. 1.5, 9.2, 9.3
- [112] Atharva Naik, Jessica Ruhan Yin, Anusha Kamath, Qianou Ma, Sherry Tong-shuang Wu, R Charles Murray, Christopher Bogart, Majd Sakr, and Carolyn P Rose. Providing tailored reflection instructions in collaborative learning using large language models. *British Journal of Educational Technology*, 56(2):531–550, 2025. 9.3
- [113] Wendy C Newstetter and Cindy E Hmelo. Distributing cognition or how they don't: An investigation of student collaborative learning. In *ICLS '96: Proceedings of the 1996 international conference on Learning sciences*. Association for the Advancement of Computing in Education (AACE), 1996. 2.2.4
- [114] Vu Nguyen, Hai Hoang Dang, Nguyen-Kha Do, and Dan-Thu TranWe. Enhancing team collaboration through integrating social interactions in a webbased development environment. *Computer Applications in Engineering Education*, 24(4): 529–545, 7 2016. ISSN 1099-0542. doi: 10.1002/cae.21729. URL <http://doi.org/10.1002/cae.21729>. 4.1.1
- [115] Omid Noroozi, Armin Weinberger, Harm JA Biemans, Martin Mulder, and Mohammad Chizari. Facilitating argumentative knowledge construction through a transactive discussion script in cscl. *Computers & Education*, 61:59–76, 2013. 5.4.4
- [116] Benjamin D Nye, Dillon Mee, and Mark G Core. Generative large language models for dialog-based tutoring: An early consideration of opportunities and concerns. In *LLM AIED*, pages 78–88, 2023. 9.1
- [117] Garrett O'Day. Improving problem solving with retrieval-based learning. Master's thesis, Purdue University, 2019. 9.1
- [118] Angela M O'Donnell. Structuring dyadic interaction through scripted cooperation. In *Cognitive perspectives on peer learning*, pages 179–196. Routledge, 2014. 2.2.1
- [119] Angela M O'Donnell and Donald F Dansereau. Scripted cooperation in student dyads: A method for analyzing and enhancing academic learning and performance. *Interaction in cooperative groups: The theoretical anatomy of group learning*, pages 120–141, 1992. 2.2.1
- [120] Fred GWC Paas and Jeroen JG Van Merriënboer. Variability of worked examples

- and transfer of geometrical problem-solving skills: A cognitive-load approach. *Journal of educational psychology*, 86(1):122, 1994. 2.1
- [121] Sherry L Piezon and Robin L Donaldson. Online groups and social loafing: Understanding student-group interactions. *Online Journal of Distance Learning Administration*, 8(4):1, 2005. 4.1.3
- [122] Leo Porter, Cynthia Bailey Lee, and Beth Simon. Halving fail rates using peer instruction: a study of four computer science courses. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 177–182. Acm, 2013. 2.2.1
- [123] Anna N Rafferty, Emma Brunskill, Thomas L Griffiths, and Patrick Shafto. Faster teaching via pomdp planning. *Cognitive science*, 40(6):1290–1332, 2016. 7.2
- [124] Taffy E Raphael, SUSAN I McMAHON, Virginia J Goatley, Jessica L Bentley, Fenice B Boyd, Laura S Pardo, and Deborah A Woodman. Research directions: Literature and discussion in the reading program. *Language Arts*, 69(1):54–61, 1992. 2.2.1
- [125] Siddharth Reddy, Sergey Levine, and Anca Dragan. Accelerating human learning with deep reinforcement learning. In *NIPS'17 Workshop: Teaching Machines, Robots, and Humans*, pages 5–9, 2017. 7.2
- [126] Alexander Renkl. *Learning from worked examples: How to prepare students for meaningful problem solving.*, page 118–130. Society for the Teaching of Psychology, 2014. 1, 2.1
- [127] Alexander Renkl. Toward an instructionally oriented theory of example-based learning. *Cognitive science*, 38(1):1–37, 2014. 1, 2.1, 6, 9.1
- [128] Alexander Renkl, Heinz Mandl, and Hans Gruber. Inert knowledge: Analyses and remedies. *Educational Psychologist*, 31(2):115–121, 1996. 5.4.1
- [129] Christian J Resick, Toshio Murase, Kenneth R Randall, and Leslie A DeChurch. Information elaboration and team performance: Examining the psychological origins and environmental contingencies. *Organizational Behavior and Human Decision Processes*, 124(2):165–176, 2014. 5.4.1
- [130] Fernando J. Rodríguez, Kimberly Michelle Price, and Kristy Elizabeth Boyer. Exploring the pair programming process: Characteristics of effective collaboration. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, Sigcse '17, pages 507–512, New York, NY, USA, 2017. Acm. ISBN 978-1-4503-4698-6. doi: 10.1145/3017680.3017748. URL <http://doi.acm.org/10.1145/3017680.3017748>. 4.1.1
- [131] J Roschelle. The construction of shared knowledge in collaborative problem solving. *Computer Supported Collaborative Learning/Springer-Verlag*, 1995. 2.2
- [132] Carolyn Penstein Rosé and Oliver Ferschke. Technology support for discussion based learning: From computer supported collaborative learning to the future of massive open online courses. *International Journal of Artificial Intelligence in Education*

*tion*, 26(2):660–678, 2016. 2.2.1

- [133] Fernando Rosell-Aguilar. Task design for audiographic conferencing: Promoting beginner oral interaction in distance language learning. *Computer assisted language learning*, 18(5):417–442, 2005. 4.2
- [134] Ron JCM Salden, Vincent Aleven, Rolf Schwonke, and Alexander Renkl. The expertise reversal effect and worked examples in tutored problem solving. *Instructional Science*, 38:289–307, 2010. 2.1.2, 9.3
- [135] Stephan Salinger, Christopher Oezbek, Karl Beecher, and Julia Schenk. Saros: An eclipse plug-in for distributed party programming. In *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering*, Chase ’10, pages 48–55, New York, NY, USA, 2010. Acm. ISBN 978-1-60558-966-4. doi: 10.1145/1833310.1833319. URL <http://doi.acm.org/10.1145/1833310.1833319>. 4.1.1
- [136] Sreecharan Sankaranarayanan. Online mob programming: Effective collaborative project-based learning. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, Sigcse ’19, page 1296, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450358903. doi: 10.1145/3287324.3293709. URL <https://doi.org/10.1145/3287324.3293709>. 5.1, 6.1.1, 8.1
- [137] Sreecharan Sankaranarayanan, Gaurav Singh Tomar, Miaomiao Wen, Akash Bharadwaj, and Carolyn Penstein Rosé. From insights to interventions: Informed design of discussion affordances for natural collaborative exchange. In *2016 AAAI Spring Symposium Series*, 2016. 5.3
- [138] Sreecharan Sankaranarayanan, Cameron Dashti, Chris Bogart, Xu Wang, Majd Sakr, and Carolyn Penstein Rosé. When optimal team formation is a choice-self-selection versus intelligent team formation strategies in a large online project-based course. In *International Conference on Artificial Intelligence in Education*, pages 518–531. Springer, 2018. 3.1, 5.3
- [139] Sreecharan Sankaranarayanan, Cameron Dashti, Chris Bogart, Majd Sakr, and Carolyn Rose. Transactivity-based team-formation as a choice: Evaluation in a large online project course. In *7th International Collective Intelligence Conference*, 2019. 3.1
- [140] Sreecharan Sankaranarayanan, Cameron Dashti, Chris Bogart, Majd Sakr, and Carolyn Rose. Transactivity-based team-formation as a choice: Evaluation in a large online project course. *7th International Collective Intelligence Conference*, Jun 2019. 2.2.1
- [141] Sreecharan Sankaranarayanan, Xu Wang, Cameron Dashti, Haokang An, Clarence Ngoh, Michael Hilton, Majd Sakr, and Carolyn Rosé. Online mob programming: Bridging the 21st century workplace and the classroom. *Proceedings of Computer-Supported Collaborative Learning* (Vol. 2.), 2019. 5.1
- [142] Sreecharan Sankaranarayanan, Xu Wang, Cameron Dashti, Marshall An, Clarence Ngoh, Michael Hilton, Majd Sakr, and Carolyn Rosé. An intelligent-agent facil-

- itated scaffold for fostering reflection in a team-based project course. In *International Conference on Artificial Intelligence in Education*, pages 252–256. Springer, 2019. 6.1.1
- [143] Sreecharan Sankaranarayanan, Xu Wang, Cameron Dashti, Marshall An, Clarence Ngoh, Michael Hilton, Majd Sakr, and Carolyn Rosé. An intelligent-agent facilitated scaffold for fostering reflection in a team-based project course. In Seiji Isotani, Eva Millán, Amy Ogan, Peter Hastings, Bruce McLaren, and Rose Luckin, editors, *Artificial Intelligence in Education*, pages 252–256, Cham, 2019. Springer International Publishing. ISBN 978-3-030-23207-8. 5.3
- [144] Sreecharan Sankaranarayanan, Siddharth Reddy Kandimalla, Sahil Hasan, Haokang An, Christopher Bogart, R Charles Murray, Michael Hilton, Majd Sakr, and Carolyn Rosé. Agent-in-the-loop: conversational agent support in service of reflection for learning during collaborative programming. In *International Conference on Artificial Intelligence in Education*, pages 273–278. Springer, 2020. 5.1.1, 5.4.1
- [145] Sreecharan Sankaranarayanan, Siddharth Reddy Kandimalla, Sahil Hasan, Haokang An, Christopher Bogart, R Charles Murray, Michael Hilton, Majd Sakr, and Carolyn Rose. Creating opportunities for transactive exchange for learning in performance-oriented team projects. *The Interdisciplinarity of the Learning Sciences, 14th International Conference of the Learning Sciences (ICLS)*, 2020. 5.4.1
- [146] Sreecharan Sankaranarayanan, Siddharth Reddy Kandimalla, Christopher Bogart, R Charles Murray, Haokang An, Michael Hilton, Majd Sakr, and Carolyn Rosé. Comparing example-based collaborative reflection to problem solving practice for learning during team-based software engineering projects. *arXiv preprint arXiv:2107.00218*, 2021. 1.2, 6.1.1, 6.5, 9.1
- [147] Sreecharan Sankaranarayanan, Siddharth Reddy Kandimalla, Christopher Bogart, R Charles Murray, Michael Hilton, Majd Sakr, and Carolyn Rosé. Combining collaborative reflection based on worked-out examples with problem-solving practice: Designing collaborative programming projects for learning at scale. In *Proceedings of the Eighth ACM Conference on Learning Scale*, pages 255–258, 2021. 1.2, 6.2.1, 6.5, 9.1
- [148] Sreecharan Sankaranarayanan, R. Charles Murray, and Carolyn Penstein Rosé. *Conversational Agents as Online Learning Tutors Supporting Collaborative Learning*, page 181–206. Education in a Competitive and Globalizing World. Nova Science Publishers, 2021. 1.3, 9.2
- [149] Sreecharan Sankaranarayanan, Siddharth Reddy Kandimalla, Christopher A Bogart, R Charles Murray, Michael Hilton, Majd F Sakr, and Carolyn P Rosé. Collaborative programming for work-relevant learning: Comparing programming practice with example-based reflection for student learning and transfer task performance. *IEEE Transactions on Learning Technologies*, 15(5):594–604, 2022. 1.2, 6.5, 9.1

- [150] Sreecharan Sankaranarayanan, Lanmingqi Ma, Siddharth Reddy Kandimalla, Ihor Markevych, Huy Nguyen, R Charles Murray, Christopher Bogart, Michael Hilton, Majd Sakr, and Carolyn Penstein Rosé. Collaborative reflection "in the flow" of programming: Designing effective collaborative learning activities in advanced computer science contexts. In *General Proceedings of the 2nd Annual Meeting of the International Society of the Learning Sciences 2022*, 2022. 8
- [151] Sreecharan Sankaranarayanan, Chellie Harrison, Sarita Kumari, Kim Larson, Robert Lemiesz, Nicolas Mesa, Ryan Mitts, Ishita Verma, and Dawn Zimmaro. Building an infrastructure for a/b experiments at scale: The challenges, opportunities, and lessons for the learning analytics community. In *LAK23 Conference Proceedings*, page 37, 2023. 9.3
- [152] Sreecharan Sankaranarayanan, Conrad Borchers, Sebastian Simon, Elham Tajik, Amine Hatun Ataş, Berkan Celik, Francesco Balzan, et al. Automating thematic analysis with multi-agent llm systems. 2025. 9.1, 9.3, 9.4
- [153] H Müge Satar and Nesrin Özdemir. The effects of synchronous cmc on speaking proficiency and anxiety: Text versus voice chat. *The Modern Language Journal*, 92(4):595–613, 2008. 4.2
- [154] Jean Scholtz and Susan Wiedenbeck. Learning second and subsequent programming languages: A problem of transfer. *International Journal of Human-Computer Interaction*, 2(1):51–72, 1990. 2.1.1, 5.4.1
- [155] Matthias Schwaighofer, Freydis Vogel, Ingo Kollar, Stefan Ufer, Anselm Strohmaier, Ilka Terwedow, Sarah Ottinger, Kristina Reiss, and Frank Fischer. How to combine collaboration scripts and heuristic worked examples to foster mathematical argumentation—when working memory matters. *International Journal of Computer-Supported Collaborative Learning*, 12(3):281–305, 2017. 2.2.3
- [156] Rolf Schwonke, Alexander Renkl, Carmen Krieg, Jörg Wittwer, Vincent Aleven, and Ron Salden. The worked-example effect: Not an artefact of lousy control conditions. *Computers in human behavior*, 25(2):258–266, 2009. 1, 1.1, 2.1
- [157] Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Nicolas Papernot, Ross Anderson, and Yarin Gal. Ai models collapse when trained on recursively generated data. *Nature*, 631(8022):755–759, 2024. 9.1, 9.3
- [158] Pooja G Sidney, Shanta Hattikudur, and Martha W Alibali. How do contrasting cases and self-explanation promote learning? evidence from fraction division. *Learning and Instruction*, 40:29–38, 2015. 2.2.3
- [159] Michelle K Smith, William B Wood, Wendy K Adams, Carl Wieman, Jennifer K Knight, Nancy Guild, and Tin Tin Su. Why peer discussion improves student performance on in-class concept questions. *Science*, 323(5910):122–124, 2009. 2.2.1
- [160] Elliot Soloway. Learning to program= learning to construct mechanisms and explanations. *Communications of the ACM*, 29(9):850–858, 1986. 2.1.1
- [161] Elliot Soloway and Kate Ehrlich. Empirical studies of programming knowledge.

*IEEE Transactions on software engineering*, Se-10(5):595–609, 1984. 2.1.1

- [162] Gerry Stahl. A decade of cscl. *International Journal of Computer-Supported Collaborative Learning*, 10:337–344, 2015. 2.2
- [163] Gerry Stahl and Kai Hakkarainen. Theories of cscl. In *International handbook of computer-supported collaborative learning*, pages 23–43. Springer, 2021. 2.2.1
- [164] Gerry Stahl, Timothy D Koschmann, and Daniel D Suthers. Cscl: An historical perspective. <https://scholarspace.manoa.hawaii.edu/server/api/core/bitstreams/99405c7b-4bd5-4492-a3f4-255483bed3b9/content>, 2006. Accessed: 2024-02-03. 2.2.1
- [165] Thomas Staibitz, Jan Renz, Christian Willems, and Christoph Meinel. Supporting social interaction and collaboration on an xmooc platform. *Proc. EDULEARN14*, pages 6667–6677, 2014. 2.2.1
- [166] Karsten Stegmann, Armin Weinberger, and Frank Fischer. Facilitating argumentative knowledge construction with computer-supported collaboration scripts. *International journal of computer-supported collaborative learning*, 2(4):421–447, 2007. 6.1.1
- [167] Karsten Stegmann, Christof Wecker, Armin Weinberger, and Frank Fischer. Collaborative argumentation and cognitive elaboration in a computer-supported collaborative learning environment. *Instructional Science*, 40:297–323, 2012. 2.2
- [168] GR Stockwell. Cmc for language learning: Examining the possibilities. In *JALT-CALL 2004 Conference, Tokiwa University, Mito, Japan*, 2004. 4.2
- [169] David Stotts, Laurie Williams, Nachiappan Nagappan, Prashant Baheti, Dennis Jen, and Anne Jackson. Virtual teaming: Experiments and experiences with distributed pair programming. In Frank Maurer and Don Wells, editors, *Extreme Programming and Agile Methods - XP/Agile Universe 2003*, pages 129–141, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-45122-8. 4.1.1
- [170] John Sweller and Graham A Cooper. The use of worked examples as a substitute for problem solving in learning algebra. *Cognition and instruction*, 2(1):59–89, 1985. 1, 2.1
- [171] John Sweller, Jeroen JG Van Merriënboer, and Fred GWC Paas. Cognitive architecture and instructional design. *Educational psychology review*, 10(3):251–296, 1998. 2.1
- [172] Gaurav Singh Tomar, Sreecharan Sankaranarayanan, and Carolyn Penstein Rosé. Intelligent conversational agents as facilitators and coordinators for group work in distributed learning environments (moocs). In *2016 AAAI Spring Symposium Series*, volume 2, page 2, 2016. 4.1
- [173] Gaurav Singh Tomar, Sreecharan Sankaranarayanan, Xu Wang, and Carolyn Penstein Rosé. Coordinating collaborative chat in massive open online courses. *arXiv preprint arXiv:1704.05543*, 2017. 4.1
- [174] Dimitra Tsovaltzi, Raluca Judele, Thomas Puhl, and Armin Weinberger. Scripts,

individual preparation and group awareness support in the service of learning in facebook: How does cscl compare to social networking sites? *Computers in Human Behavior*, 53:577–592, 2015. 5.1.4

- [175] Dimitra Tsovaltzi, Armin Weinberger, Lara Schmitt, Henrik Bellhäuser, Adrienne Müller, Johannes Konert, René Röpke, Elise Eshuis, Tessa Eysink, Melanie Erkens, et al. Group formation in the digital age: Relevant characteristics, their diagnosis, and combination for productive collaboration. *Proceedings of Computer-Supported Collaborative Learning* (Vol. 2.), 2019. 3.1, 5.2.1
- [176] Juhani E Tuovinen and John Sweller. A comparison of cognitive load associated with discovery learning and worked examples. *Journal of educational psychology*, 91(2):334, 1999. 2.2.3
- [177] Tomoyuki Urai, Takeshi Umezawa, and Noritaka Osawa. Enhancements to support functions of distributed pair programming based on action analysis. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, pages 177–182. Acm, 2015. 4.1.1
- [178] Carla Van Boxtel, Jos Van der Linden, and Gellof Kanselaar. Collaborative learning tasks and the elaboration of conceptual knowledge. *Learning and instruction*, 10(4):311–330, 2000. 2.2, 5.4.1
- [179] Carla Van Boxtel, Jos van der Linden, Erik Roelofs, and Gijsbert Erkens. Collaborative concept mapping: Provoking and supporting meaningful discourse. *Theory into practice*, 41(1):40–46, 2002. 2.2, 5.4.1
- [180] Antoine Van den Beemt, Miles MacLeod, Jan Van der Veen, Anne Van de Ven, Sophie Van Baalen, Renate Klaassen, and Mieke Boon. Interdisciplinary engineering education: A review of vision, teaching, and support. *Journal of engineering education*, 109(3):508–555, 2020. 2.2
- [181] Tamara Van Gog, Fred Paas, and Jeroen JG Van Merriënboer. Effects of process-oriented worked examples on troubleshooting transfer performance. *Learning and Instruction*, 16(2):154–164, 2006. 1, 2.1
- [182] Tamara Van Gog, Liesbeth Kester, and Fred Paas. Effects of worked examples, example-problem, and problem-example pairs on novices' learning. *Contemporary Educational Psychology*, 36(3):212–218, 2011. 2.1.2
- [183] Jeroen JG Van Merriënboer and Fred GWC Paas. Automation and schema acquisition in learning elementary computer programming: Implications for the design of practice. *Computers in human behavior*, 6(3):273–289, 1990. 2.1.1
- [184] Camilo Vieira, Junchao Yan, and Alejandra J Magana. Exploring design characteristics of worked examples to support programming and algorithm design. *Journal of Computational Science Education*, 6(1):2–15, 2015. 2.1
- [185] Rosanna Vitiello, Soham D Tiwari, R Charles Murray, and Carolyn Rosé. Traveling bazaar: Portable support for face-to-face collaboration. In *International Collaboration toward Educational Innovation for All: International Society of the Learning*

- [186] Lev Semenovich Vygotsky. *Mind in society: The development of higher psychological processes*. Harvard university press, 1980. 2.2.1
- [187] Lev Semenovich Vygotsky and Michael Cole. *Mind in society: Development of higher psychological processes*. Harvard university press, 1978. 2.2.1
- [188] Erin Walker, Nikol Rummel, and Kenneth R Koedinger. Beyond explicit feedback: new directions in adaptive collaborative learning support. In *Proceedings of the 9th international conference on Computer supported collaborative learning-Volume 1*, pages 552–556, 2009. 5.3
- [189] Pengcheng Wang, Jonathan P Rowe, Wookhee Min, Bradford W Mott, and James C Lester. Interactive narrative personalization with deep reinforcement learning. In *Ijcai*, pages 3852–3858, 2017. 7.2
- [190] Xu Wang, Miaomiao Wen, and Carolyn Rosé. Contrasting explicit and implicit support for transactive exchange in team oriented project based learning. In *2017 Proceedings of the Computer-Supported Collaborative Learning Conference.*, 2017. 4.1, 5.3
- [191] Yansen Wang, R Charles Murray, Haogang Bao, and Carolyn Rose. Agent-based dynamic collaboration support in a smart office space. In *Proceedings of the 21th annual meeting of the special interest group on discourse and dialogue*, pages 257–260, 2020. 1.5, 9.2
- [192] Noreen M Webb and Annemarie Sullivan Palincsar. *Group processes in the classroom*. Prentice Hall International, 1996. 2.2.1
- [193] Armin Weinberger. *Scripts for computer-supported collaborative learning*. PhD thesis, lmu, 2003. 2.2.2
- [194] Armin Weinberger. Principles of transactive computer-supported collaboration scripts. *Nordic Journal of Digital Literacy*, 6(03):189–202, 2011. 7.5.1
- [195] Armin Weinberger, Miika Marttunen, Leena Laurinen, and Karsten Stegmann. Inducing socio-cognitive conflict in finnish and german groups of online learners by cscl script. *International Journal of Computer-Supported Collaborative Learning*, 8 (3):333–349, 2013. 7.5.1
- [196] Miaomiao Wen. *Investigating virtual teams in massive open online courses: deliberation-based virtual team formation, discussion mining and support*. PhD thesis, Carnegie Mellon University, 2016. 2.2.1, 5.3
- [197] Miaomiao Wen, Keith Maki, Xu Wang, Steven Dow, James D Herbsleb, and Carolyn Penstein Rosé. Transactivity as a predictor of future collaborative knowledge integration in team-based learning in online courses. In *Edm*, pages 533–538, 2016. 2.2.1, 5.3.1
- [198] Miaomiao Wen, Keith Maki, Steven Dow, James D. Herbsleb, and Carolyn Rose. Supporting virtual team formation through community-wide deliberation. *Proc. ACM Hum.-Comput. Interact.*, 1(Cscw):109:1–109:19, December 2017. ISSN 2573-

0142. doi: 10.1145/3134744. URL <http://doi.acm.org/10.1145/3134744>. 2.2.1, 5.3
- [199] Alexander Wilson. Mob programming-what works, what doesn't. In *International Conference on Agile Software Development*, pages 319–325. Springer, 2015. 2.2.1, 2.2.2, 4.1, 6.1.1, 8.1
- [200] Zheyuan Zhang, Daniel Zhang-Li, Jifan Yu, Linlu Gong, Jinchang Zhou, Zhanxin Hao, Jianxiao Jiang, Jie Cao, Huiqin Liu, Zhiyuan Liu, et al. Simulating classroom education with llm-empowered agents. *arXiv preprint arXiv:2406.19226*, 2024. 9.3
- [201] Rui Zhi, Thomas W Price, Samiha Marwan, Alexandra Milliken, Tiffany Barnes, and Min Chi. Exploring the impact of worked examples in a novice programming environment. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 98–104, 2019. 1, 2.1, 2.1.1
- [202] Xuhui Zhou, Zhe Su, Tiwalayo Eisape, Hyunwoo Kim, and Maarten Sap. Is this the real life? is this just fantasy? the misleading success of simulating social interactions with llms. *arXiv preprint arXiv:2403.05020*, 2024. 9.3
- [203] Hildert Zoethout, Renate Wesselink, Piety Runhaar, and Martin Mulder. Using transactivity to understand emergence of team learning. *Small group research*, 48(2):190–214, 2017. 7.5.1
- [204] Woody Zuill. Mob programming: A whole team approach. In *Agile 2014 Conference, Orlando, Florida*, 2016. 2.2.1, 2.2.2, 4.1, 6.1.1, 8.1

## **Acknowledgments**

Placeholder.