# Alienation and Cognitive Delegation:
# The Phenomenology of Learning in AI-Assisted Programming Education

by
benjamin cassidy

A project submitted in partial fulfillment of the
requirements for the degree of

Master of Science in Computer Science

Department of Computer Science and Cybersecurity
College of Sciences
Metropolitan State University
St. Paul, Minnesota
May 5, 2026

## Disclaimer

The author declares that they are the sole author of this thesis and that they have not used any sources other than those listed in the references or cited along with the text. The author further declares that they have not submitted this work to any other institution for obtaining a degree.

## Abstract

As artificial intelligence tools become ubiquitous in programming education, fundamental questions emerge about how students develop expertise when AI can automate the cognitive labor through which learning occurs. This thesis applies Marx's theory of alienation to AI-assisted programming education, developing a theoretical framework of "pedagogical alienation" to analyze how automation transforms students' relationships to their intellectual work.

Programming expertise requires pattern thinking—the ability to recognize recurring problem structures, select appropriate design patterns, and synthesize elegant solutions. This craft knowledge develops through sustained practice and productive struggle. When AI automates the productive transformation of problems into solutions, it prevents the formative transformation of students into experts. Educational labor has a distinctive property: the worker and the product are the same entity. When AI performs the cognitive work that should produce understanding, students receive functioning code while remaining unchanged.

This research examines how four types of pedagogical alienation manifest in students learning design patterns under varying AI assistance conditions. Using mixed methods embedded within an actual Object-Oriented Design course, the study investigates whether students experience product alienation (artifacts without understanding), process alienation (reduced to prompting rather than problem-solving), species-being alienation (failure to develop metacognitive capacities), and social alienation (preferring AI over peer collaboration).

The findings reveal that AI assistance creates meta-alienation: students experience dependency as empowerment, reporting high confidence while demonstrating poor independent capability. This research contributes a theoretical framework for analyzing AI's epistemological and ethical implications in education, challenging effectiveness-focused approaches that ignore how technological mediation fundamentally alters the learning process itself.


**Keywords:** Pedagogical alienation, Artificial intelligence in education, Marx alienation theory, Programming pedagogy, Pattern thinking, Critical pedagogy, educational technology ethics

## Dedication

To my wife for enduring this. To my students—past, present, future—who make it matter.

“In the manufacturing workshop and in craft labor, tools serve the worker; in the factory, the worker serves the machine. In one case, he moves the means of labor; in the other, his job is to follow their movement.”

— Karl Marx, *Capital*

## Acknowledgements

I am grateful first to my thesis advisor, Professor Jigang Liu, whose guidance shaped this project at every stage. He consistently allowed this work to be my own while steering me toward greater rigor and clarity—asking the questions that pushed me to think more carefully and offering the kind of mentorship that made a difficult project feel possible.

I would also like to thank my committee members, Professor Thanaa Ghanem and Professor Cheng Thao, for their participation, their careful reading, and their willingness to engage seriously with ideas that sit at the intersection of critical theory and computer science education. Their input strengthened this thesis considerably.

I am grateful to the students of ICS 372, who did not simply provide data — they thought carefully about their own learning, documented their uncertainties with honesty, and in many cases arrived at insights about AI and intellectual development that this thesis could not have reached without them. Whatever this research understands about pedagogical alienation, it learned first from them.

Most importantly, I thank my wife, who has endured years of digressions on alienation, commodity fetishism, and the labor theory of value, who bore with me through the many false starts on this project, who endured hours upon hours of solitary research and writing, and who, during the final days of writing, would not be drawn into discussion until after the click of the final keystroke.

# Table of Contents

# List of Tables

# List of Figures

# 1 The Transformation of Programming Education

Is the mind that codes evaporating, by degrees, by design? Will we continue to see the lone figure hunched in monitor-blue darkness, wrestling problems into submission long into the night in a caffeine-buzzed flow state, driven by sheer force of intellect and an insatiable curiosity, refusing to surrender until the pattern yields to will. Or is that same coder now typing a prompt and watching the machine code itself—no longer producing but consuming, no longer creating but channeling artificial thought. Presently necessary, but soon obsolete.

The mind that codes—what is it, exactly? More than a repository of syntax, more than a connector of frameworks—it thinks in systems. Essential to the work of a coder is the ability to detect the patterns that emerge in these systems, to apply these patterns in unfamiliar domains, and to synthesize patterns into functional code that solves real problems.

The ability to detect patterns is perhaps the most mysterious aspect of programming expertise. An experienced programmer scans the source code in an unfamiliar codebase, spotting the established design patterns and the violations thereof. They map the architecture like surveying familiar territory: *observer patterns* watching the data model, *facades* hiding third-party complexity, *state patterns* governing transitions. This skill cannot be taught, but rather it must be trained. One must see these patterns and apply them in practice many times before the detection is a natural part of their process. We programmers see in code the underlying structures regardless of the actual implementation. In this sometimes chaotic mess, we tease out these structures and add them to our mental map. Like a sculptor looking at a block of marble and seeing a man, we look at a system and see the patterns.

Knowing when to use patterns is the true test of mastery. Having seen the *command pattern* in the past, the master programmer understands that it solves the problem of creating a

stack of edit objects that can be undone and that it can also be used to delegate work to a thread pool. The requisite cognitive leap is going from an understanding of the shape of the code that implements the pattern to an internalization of the abstract principle. Not just the how, but the *why*. Developers who know design patterns never start a program from scratch. They have much of it written in their heads before they open their text editor. When implementing a user interface, they do not have to come up with how to handle user input, they implement the *observer pattern* without even needing to name it. An artisan can take patterns developed for a specific domain and apply them in far-ranging contexts. The Model-View-Controller pattern, developed for graphical user interfaces, can be used to simplify the implementation of an anti-lock braking system in an automobile.

The highest level of pattern thinking is synthesis—the artful combination of multiple patterns into elegant, robust, and bespoke solutions. The craft of programming emerges at this juncture—for programmers are craftspeople who employ a variety of patterns in service of work that has internal structure and consistency. Consider a real-time messaging system: the artisan might combine an *observer* for receiving event notifications, with the *strategy pattern* for different message types, a *factory* for creating connections, and a *decorator* for handling encryption. The task is to discern what is the best combination of these primitive tools. We ought to understand how our decisions affect the evolution of the software and when the patterns should be broken—then doing so judiciously. The requirement that there should only be a single instance of an object in the system does not always mean that a *singleton* is the right choice. True craftsmanship requires the ability to see how a combination chosen for economy at this stage of a project, will lead to unnecessary complications in the future. This is the creative act at the heart

of programming—not just knowing the patterns, but crafting them into solutions that feel inevitable.

This, then, is what evaporates when we optimize away the struggle. Each semester, the evaporation accelerates as every new cohort of students relies more on generative artificial intelligence (AI). They perform less cognitive archaeology to reveal for themselves the patterns in code. They never learn to synthesize elegant solutions from primitive components. Indeed, they may never see the components themselves, moving directly from prompt to submission. What was once the slow cultivation of expertise, hard won through devising solutions to increasingly difficult problems, is now only the repackaging of automated answers. Our students are no longer producers, but consumers.

The implications extend far beyond the classroom. We risk creating a generation of programmers with enough knowledge to get a program compiling and running, but no insight into its inner workings. Every line of code they write, just another brick in an inscrutable black box. They can execute, but cannot architect. They implement, but will never innovate. This is not merely pedagogical inefficiency—it is cognitive atrophy with civilizational consequences.

Three questions emerge from this milieu—questions at once pedagogical and philosophical. On the pedagogical side, they can be stated plainly, as matters of educational practice and assessment:

1. RQ1: How does AI assistance affect students' ability to detect design patterns in unfamiliar code?
2. RQ2: How does AI assistance influence students' capacity to apply learned patterns across different domains?

3. RQ3: How does AI assistance impact students' skill in synthesizing multiple patterns into elegant solutions?

These are the practical concerns of pedagogy. But behind them lie older and more fundamental questions. These same questions, when refracted through the lens of philosophy, reveal their deeper significance. What becomes of recognition when the very act of recognition is outsourced to a machine? What becomes of transfer when learning itself is reduced to replication? And what becomes of synthesis when creation collapses into consumption?

This thesis therefore proceeds in a dual register—empirical and philosophical. The computer science educator asks what students can or cannot do when assisted by AI; the philosopher asks what it means for the cultivation of mastery when the labor of learning is automated away.

To investigate these questions, we need a rigorous theoretical framework to not only examine *what* students are learning, but *how* the process of learning is being transformed. Karl Marx's theory of alienation, developed to explain the dehumanizing effects of industrial labor, provides precisely such a framework. It will guide us as we analyze how AI assistance may be reshaping a student's relationship to their intellectual work.

Marx identified four types of alienation [1]:

1. *Product Alienation (Alienation from the Product of Labor)*: Workers are separated from what they create. The output of their labor, their products, are owned by someone else and appear foreign to them. Once the product is created the worker has no control over what is done with the product, and in many cases may have no understanding of how their contribution manifests in the final product.

2. *Process Alienation (Alienation from the Act of Production)*: Workers have no control over *how* they work. Instead of work being a creative and intellectual self-directed endeavor, it is repetitive, mechanical, and dictated to them by others. Work is something done *to* the worker, rather than something done *by* the worker.
3. *Species-Being Alienation (Alienation from Human Essence)*: Humans are naturally creative, conscious, curious beings who transform their environment through work that has a specific purpose. Under capitalism, their work becomes a means for survival alone and contains none of the creativity that would allow a worker to achieve their full human potential.
4. *Social Alienation (Alienation from Fellow Workers)*: Workers relate to one another, not as cooperating members of a human community, but as competitors in a market. Social relationships become defined by economic roles and private property rather than genuine human connection.

Education may not be considered labor in the traditional sense, but learning is fundamentally a form of human work. Students invest time, effort, and intellectual energy to transform problems, questions, and data into genuine understanding and competency—they produce new knowledge, skills, and capabilities through their cognitive labor. In computer science education specifically, students engage in the same essential activities as professional programmers: requirements analysis, solution architecting, coding, and debugging. The scale is different, but the work is the same.

Still, the education context requires us to adapt Marx's framework. While workers in the marketplace relate to bosses and private property, students increasingly relate to their education through the mediation of an AI that can carry much of the burden of that intellectual labor. The

four types of alienation take on new meaning when the mediating force is not an owner, but an AI assistant that can analyze, summarize, adapt, or even create on the student's behalf.

Taking this into account, we describe the four types of alienation in an education context as follows:

1. *Product Alienation (Alienation from Understanding)*: The primary product of education is understanding. When a student relies on an AI assistant to produce their coursework, they lose the total understanding they set out to acquire. In terms of programming education this may manifest in a student who cannot explain how their code works or why it doesn't work.
2. *Process Alienation (Alienation from the Process of Problem-Solving)*: In order to produce new understanding, a student must be faced with questions or tasks beyond the scope of their current understanding. The AI assistant enables a student to move from problem to solution without participating in the task other than to type a prompt. A programming student may produce working code, but they cannot replicate the intellectual process that led to the solution—they've skipped the essential work of problem-solving.
3. *Species-Being Alienation (Alienation from Intellectual and Creative Potential)*: Learning is a fundamentally creative and transformative process where students encounter new questions, cultivate unexpected interests, and develop novel approaches to problems. This process of intellectual growth and self-actualization is central to the development of an integrated and whole person. With AI assistance, learning may be reduced to a set of input-output transactions, depriving students of the sense of creativity, curiosity, and intellectual agency that inspired them to seek an education. In programming education, this manifests when students miss the satisfaction of breakthrough moments—the hard-

won understanding that comes after struggling to debug code, or the creative insight that emerges when synthesizing multiple concepts into an elegant solution.

4. *Social Alienation (Alienation from Community of Learners)*: Education happens in classrooms with other students who bring their own experiences and insights. Through discussion and collaboration, students enrich their own educational experience, deepen their understanding, and participate in a community of fellow learners interested in the same topic. When an AI assistant is engaged, students no longer have the need to seek the input of others. In a programming class, students would miss seeing how classmates approached the same problem differently, or experiencing that moment of clarity that comes from explaining their code to a peer—realizing their own understanding through the act of teaching.

## 1.1 Research Significance and Contribution

As AI tools become ubiquitous in computer science education, fundamental questions emerge about how students develop technical expertise. Much attention has focused on how well AI performs in comparison to students and AI's potential to enhance learning efficiency. This research for the most part takes a market-oriented approach, evaluating outcomes in terms of production speed and quality. Fewer researchers are exploring how the use of AI tools during the learning process transforms the relationship between the learner and subject-matter—their development of technical mastery. We remain uncertain whether we are educating programmers capable of creating the AI tools they have come to depend upon.

This thesis addresses this gap by applying Marx's theory of alienation to AI-assisted computer science education, extending pedagogical alienation theory to technical skills development. Rather than focusing solely on performance metrics, I examine how students

experience and relate to their own learning process when AI mediates their engagement with complex technical subjects. This research identifies novel pathways to pedagogical alienation that emerge when AI tools are used; this is distinct from traditional educational shortcuts or technological aids.

By analyzing *pattern thinking* as a form of creative labor analogous to Marx's conception of craft work, this thesis bridges critical theory and computer science education research. This cross-disciplinary approach brings philosophical depth to educational technology questions while grounding abstract theoretical concepts in empirical classroom data, providing new frameworks for understanding skill development in technical disciplines.

The urgent questions this thesis seeks to answer are facing computer science educators the world over. We have students using AI tools regardless of any external restrictions we may place on such usage. This requires us to understand not just the effects in terms of output, but also in terms of our students' understanding, so that we can tailor the way that we teach to ensure our students are still learning. The onus is on us to educate well regardless of the tools available, and we cannot do so if we don't know how the education experience has changed.

This challenge extends beyond individual classroom decisions to institutional policy-making about AI integration. Educators need evidence-based guidance on when AI assistance enhances learning versus when it undermines skill development, especially for complex technical competencies that require deep understanding rather than just correct outputs. Administrators need concrete recommendations for policy development, faculty need classroom implementation strategies, and curriculum committees need criteria for evaluating AI integration across different courses and learning objectives.

To that end, this research shows, with quantitative and qualitative data, how AI use is altering the educational environment in technical coursework. In the process I develop a new framework for adopting AI in pedagogy that distinguishes between productive and counterproductive AI integration based on learning objectives and skill development stages. This framework moves beyond the binary choice between permission or prohibition of AI tools, instead providing nuanced guidance for different types of technical learning tasks. By focusing specifically on *pattern thinking* development, this research offers actionable insights that educators can immediately apply to course design, assignment creation, and assessment strategies in ways that preserve educational rigor while embracing technological innovation.

This research employs a novel experimental design that uses individual students as their own control group by comparing their performance across different AI assistance conditions. Students receive instruction on individual design patterns and then complete assignments on the covered material under varying AI constraints—required, prohibited, optional, and critical analysis conditions. Their pattern recognition and application skills are then assessed through in-person, paper-based examinations.

Throughout the study, students provide qualitative feedback about their learning process and experiences through weekly reflection journals. This approach captures not only *what* students accomplish and *how* they complete assignments, but more importantly for this research, *why* they make specific choices during problem solving. Such process-oriented questions are often unexplored in computer science education research, making this methodological approach particularly valuable for understanding the relationship between tool usage and learning development.

By embedding this research within actual coursework rather than artificial laboratory conditions, this methodology captures authentic learning experiences. Students engage with genuine academic tasks that carry real stakes for their grades and learning outcomes, eliminating the artificial constraints that often limit the generalizability of educational research. This approach ensures that findings reflect actual classroom dynamics rather than simulated environments. While this introduces potential confounding variables typical of field research, it provides results directly applicable to real educational contexts where AI integration decisions must be made.

Computer science faces unique stakes in AI integration because our graduates create these very tools. We must maintain the educational excellence that produced today's AI's breakthroughs while preserving our students' capacity for tomorrow's innovations. Other technical disciplines—engineering, mathematics, data science—face similar challenges in maintaining rigorous skill development when AI can perform many traditional learning tasks.

Beyond STEM fields, AI integration in education raises fundamental questions about learning, creativity, and intellectual development that transcend disciplinary boundaries. While specific manifestations may differ across humanities, medicine, or social science, the underlying concerns about pedagogical alienation and authentic skill development remain constant. This research provides a theoretical framework and methodological approach that other disciplines can adapt to examine AI's impact on their own unique learning objectives and professional preparation requirements.

## 1.2 Thesis Structure and Overview

This thesis builds its case in three interconnected moves. First, I establish the theoretical foundation by demonstrating how Marx's alienation theory can be extended to educational

contexts, creating a new analytical framework for understanding how AI transforms the relationship between a student and their learning (Chapters 2-4). Second, I test this framework empirically through controlled classroom experiments that isolate the effect of AI on different aspects of the development of pattern thinking, providing concrete evidence of pedagogical alienation in action (Chapters 5-6). Finally, I synthesize these findings to argue for fundamental changes in how we approach programming education and educational technology more broadly (Chapters 7-8).

**Chapter 2** situates this inquiry within existing research on programming education and AI-assisted learning, while **Chapter 3** examines related empirical studies to establish what remains unknown about AI's effects on conceptual learning. **Chapter 4** then develops the theoretical innovation at the heart of this work—the framework of pedagogical alienation—by systematically extending Marx's four types of alienation to educational contexts and operationalizing them for empirical investigation.

The empirical core of the thesis unfolds in **Chapters 5 and 6**. **Chapter 5** details the mixed-methods approach used to study students learning design patterns under four different AI conditions: required assistance, prohibited assistance, optional assistance, and critical evaluation of AI-generated solutions. **Chapter 6** presents findings organized around each type of pedagogical alienation, revealing how different forms of AI mediation affect students' relationship to their intellectual work in distinct and measurable ways.

**Chapter 7** synthesizes these findings to argue that current approaches to AI in programming education risk creating a generation of technically competent but intellectually dependent programmers. The chapter proposes alternative pedagogical approaches that preserve the craft elements of programming while leveraging AI's benefits. **Chapter 8** concludes by

reflecting on the broader implications for educational technology and the future of expertise development in an increasingly automated world.

# 2 Literature Review: Learning, Patterns, and Alienation

Programming expertise develops through practice, not instruction. Students must learn to see patterns in code, apply solutions across contexts, and synthesize elegant architectures from primitive components. This transformation—from novice who sees only syntax to expert who thinks in abstractions—occurs through sustained engagement with difficult problems. When AI can automate this cognitive labor, what becomes of the learning process itself?

This chapter examines three bodies of scholarship essential for understanding AI's impact on programming education. Research on design patterns pedagogy reveals how pattern thinking develops through repeated practice in varied contexts, not through memorization of definitions. Cognitive science demonstrates that difficulty serves as a learning mechanism—productive struggle creates durable understanding while ease of performance produces superficial capability. Critical pedagogy provides frameworks for analyzing how technological mediation transforms students' relationships to their intellectual work, though these frameworks were developed before AI existed in classrooms.

These literatures have never intersected. Programming education research documents what works but lacks critical frameworks for examining how automation disrupts learning. Cognitive science reveals the mechanisms through which expertise develops but does not analyze what happens when those mechanisms are automated away. Critical pedagogy examines educational alienation but has not addressed AI's specific impact on cognitive labor. The gap is urgent: no research systematically analyzes how AI automation creates pedagogical alienation in programming education. This chapter establishes the foundation for that analysis.

## 2.1 Object-Oriented Programming Pedagogy and Design Patterns

The practice of programming involves thinking in two distinct modes—abstract and concrete—and the cognitive labor of translating between them. A programmer must first create a mental model (*abstract*) of the problem and then convert that model into functioning code (*concrete*). Object-oriented programming intensified these cognitive challenges by introducing additional layers of abstractions—objects, classes, inheritance, polymorphism—atop procedural programming concepts [2]. This dual-processing presents fundamental pedagogical challenges: abstractions resist clear communication, and it is difficult to assess whether students have internalized the intended conceptual structures [3].

### 2.1.1 The Challenge of Teaching Abstractions

Patterns emerged in programming pedagogy precisely to address this abstraction challenge. Research on programming expertise established that expert programmers do not approach each problem as novel—instead, they recognize recurring problem structures and deploy reusable solution templates. Soloway and Ehrlich's foundational work on "programming plans" demonstrated that experts organize their knowledge around stereotypical code fragments that achieve common goals: reading input, accumulating results, searching collections [4]. These plans function as cognitive chunks, allowing experts to work at higher levels of abstraction rather than reasoning about individual statements. Novice programmers lack these plans, forcing them to construct solutions from first principles each time—a cognitively expensive process that limits their ability to tackle complex problems.

Patterns formalize this expert knowledge, making it accessible to learners. As Gamma et al. argued in their foundational *Design Patterns* text, patterns serve three critical pedagogical functions: they provide a shared vocabulary for discussing design decisions, they capture expert

knowledge about what works in practice, and they facilitate design reuse across contexts [5]. Rather than memorizing syntax or language features, students learning through patterns acquire transferable problem-solving strategies. A student who understands the Iterator pattern, for instance, has not merely learned one code structure—they have internalized a general approach to sequential access that transfers across different data structures, languages, and problem domains.

This emphasis on transfer distinguishes pattern-based pedagogy from earlier approaches. Astrachan et al. observed that traditional "divide and conquer" or "top-down design" instruction left students unable to design and implement programs even after multiple semesters [6]. Patterns provide the missing link: reusable mental models that scaffold the transition from novice to expert thinking. By building students' design vocabulary incrementally, pattern pedagogy aims to replicate the gradual accumulation of programming knowledge that experts develop through years of practice. The pedagogical challenge, then, is not merely teaching students *what* patterns are, but helping them develop the recognition, selection, and adaptation skills that characterize expert pattern use.

Early computer science education researchers found that students struggled with learning abstractions and would reason backwards from the goal of the program to a solution through the process of coding [7]. Students lack a mental schema that they could use to plan the software before writing it and struggled to convert high-level requirements into programming language statements. Object-oriented programming was seen as a way to teach programming through abstractions and thus give students the tools to create the mental schema before writing any code.

Wallingford created a paradigm for teaching object-oriented programming through the use of *programming patterns*. Initially he identified only three patterns for use in his course:

*View* for decoupling an object's state from its presentation, *State* for providing suitable access to an object's data, and *Decorator* for adding additional functionality without modifying an object. These patterns were useful in that they were both simple enough for novice programmers to grasp yet concrete enough to "provide meaningful support for writing implementation code" [7]. Though some students did have initial resistance to using the patterns—thinking that it stifled creativity—most reported that the patterns were helpful in completing their assignments as it gave them tools with which to build programs.

### 2.1.2 Programming Patterns and Process Scaffolding

Proulx noticed that students understood syntax and constructs, but struggled to begin writing a program because they did not know where to begin [8]. During the courses, students were presented with lessons on things like loops and control flow statements, but could not structure a working program. To address this problem and to help students learn programming, Proulx identified certain *programming* patterns. These patterns are generalized concepts, distinct from *design* patterns, that apply to the process of programming, not a programming paradigm. The purpose of these patterns is to guide students from the blank text editor to a working program.

Proulx defines ten patterns for common programming tasks: Name Use Patterns, Reading Data Pattern, Read-Process-Write Pattern, Encapsulation Pattern, Repetition Patterns, Selection Patterns, Traversal Patterns, Cumulative Results Pattern, Conversion Patterns, and Indirect Reference Patterns. It would be useful to explore two of these in-depth to understand the value of this pedagogical approach.

The Name Use Patterns helps students with the first step in constructing their solutions—understanding what needs to be built. All programming languages have 'identifiers' that

reference different components in a program. Students were taught to begin by naming all the data (variables, constants, classes, etc.) and behavior (functions) the program needed, along with any type information. With this definition in hand, a student can now conceptualize the various parts of the program, what it does, and what data it needs. As we can see, this pattern is paradigm agnostic and defines a process pattern that is useful regardless of the task a student works on.

The Cumulative Results Pattern is a composite of the other patterns. It helps to solve the programming task of traversing a collection, processing the data, and returning a composite result based on some accumulator. This requires students to make multiple "independent and interdependent decisions when designing solutions to these types of problems" [8]. This pattern was presented to students as a sequence of steps each of which applied one or more of the other patterns. By seeing how the patterns could be combined to solve multi-step problems, students understood how to make other combinations of the patterns to solve problems they might encounter.

### 2.1.3 Design Patterns Pedagogy in CS Education

Object-oriented programming provided new ways of creating abstractions in code. Thus, it led to an evolution in the design and architecture of computer programs. In 1994, Gamma, et al. [5] introduced the concept of design patterns, naming 23 specific patterns and introducing best-practices such as "programming to interface" and "composition over inheritance." This text has come to be known colloquially as the Gang of Four (GoF). Design patterns are generalized reusable patterns that can be applied in various contexts to solve common problems. They are defined by a four-part schema 1) a **name** which provides a common language to refer to the pattern, 2) a **problem** definition indicating where the pattern is meant to be deployed, 3) the

**solution** exemplifying the manner in which the **problem** is solved, and 4) the **consequences** of using the patterns in terms of any trade-offs that are made when using the pattern.

However, as Astrachan et al. observed, this text assumes expert-level knowledge making it inaccessible to novice programmers—necessitating pedagogical approaches specifically designed for students [6]. Teaching these patterns effectively requires more than simple memorization of pattern structures; students must develop pattern recognition skills, learn to select appropriate patterns for given problems, and practice applying patterns across different contexts. It's not always clear when a pattern would be applicable. This requires students to develop a mental model of a program with detail sufficient enough that they can "see" the patterns emerge while they code.

At the time of their writing, Astrachan et al. noted that most approaches to teaching programming involved either teaching students with a *"divide and conquer"* or *"top-down design"* approach. After one or two semesters learning from books that employed this method students were not adept at designing and implementing programs. Instead, they advocated an approach which focuses on increasing a student's design vocabulary by introducing them to common programming patterns and idioms.

The first pattern taught to students is the *Iterator* pattern. They required students to use an iterator to read the words from a text file and count the total number of words and characters. They then required the students to apply the *Iterator* pattern in other tasks. Because they use the same member function names (first(), isDone(), and next()) in all applications of the *Iterator* pattern, students were able to make use of the pattern in other contexts with little difficulty. When they taught the *Observer* pattern they showed the student how it worked and then tasked them with applying the pattern in a simulation game. Instead of having the students write

observers (i.e. using interfaces named Observer and Subject), the students created a *Character* class which would then notify the other components in the simulation when appropriate. This taught the students to generalize the concept of observation instead of thinking only in terms of the design pattern specification.

Eigler et al. [9] conducted a systematic literature review identifying 12 educational tools designed specifically to support design pattern learning. Their analysis reveals that effective pattern education tools emphasize active learning processes: 83% of the tools support pattern detection (recognizing patterns in existing code), 83% support pattern application (applying patterns to solve problems), and 67% support pattern selection (choosing appropriate patterns for given contexts). Notably, most tools employ interactive, scaffolded approaches where students receive immediate feedback and engage in iterative problem-solving.

### 2.1.4 Contemporary Innovations

One of the tools they highlight is called DEPTHS (Design Patterns Teaching Help System) [10]. DEPTHS is a comprehensive learning environment which "integrates a Learning Management System, a software modeling tool, diverse collaboration tools and relevant online repositories of DPs." Learning with DEPTHS is a process involving collaboration with peers, software modeling, implementation, and critical evaluation. Students first present their proposed solution to their peers for feedback and provide feedback on the proposals of other students. They then use the software tool to search for previous solutions to similar problems which gives them an opportunity to understand how to apply a design pattern. Once they understand the problem context and pattern based solutions, they implement their deliverable then present it to their peers for another phase of feedback and critical evaluation. Through this process, students actively engage their critical thinking capacities in individual and social contexts to reinforce the

abstract concepts they are learning. Although the results of the study with DEPTHS does not have statistical power due to a small sample size, feedback from students indicates that they found the tool to be effective in helping them learn design patterns.

Recent work on teaching design patterns has suggested the use of video games to introduce and reinforce these concepts [11] [12]. In [12] students played an interactive Serious Video Game. They were able to select among three design patterns topics. The video game world contained two houses: a learning house and a question house. Students learned about design patterns in the learning house and answered questions in the question house. Students in the study were separated into experiment and control groups. Of those in the experiment group, 100% could identify at least three design patterns after playing the game. Furthermore, the students expressed a higher level of motivation and satisfaction with the course.

Silva, et al. used a popular video game as an exemplar for the application of design patterns, comparing students using the exemplar against a group of students receiving traditional instruction [11]. The students were presented a lecture about design patterns and then given assignments to apply those patterns. Students using the exemplar were tasked with designing the architecture of the video game using the design patterns they learned about. A larger portion of the students in the exemplar group found the instruction using the video game as an example as "very helpful" vs the group taught with traditional models. The authors conclude that the use of real-world examples helps the students concretize what they have learned by applying it in a scenario they understand.

These pedagogical innovations—from Wallingford's early patterns to DEPTHS' collaborative environments to game-based exemplars—share a common assumption: students must actively engage with programming challenges, working through multiple attempts and

receiving feedback. The question examined in this thesis is what happens when AI assistance fundamentally alters this engagement by automating the cognitive work these approaches were designed to preserve.

**Table 2.1: Pedagogical Approaches to Pattern Learning in CS Education**

| Scholar(s) | Approach | Required Student Activity | Learning Outcome | Vulnerability to AI Substitution |
|---|---|---|---|---|
| **Programming Patterns** | | | | |
| Wallingford (1996) | Three foundational OOP patterns (*View*, *State*, *Decorator*) introduced to novice programmers | Applying patterns to complete assignments; working through initial resistance | Development of pre-coding mental schema; ability to plan structure before writing | AI can generate pattern-conforming code without students constructing the schema |
| Proulx (2000) | Ten process-oriented programming patterns (e.g., *Name Use*, *Cumulative Results*) to scaffold the blank-page-to-working-program transition | Naming components, sequencing decisions, combining patterns for multi-step problems | Understanding of program structure and compositional reasoning | AI collapses the sequencing process into a single prompt, bypassing each decision point |
| **Design Patterns Pedagogy** | | | | |
| Gamma et al. (1994); Astrachan et al. (1998) | Design vocabulary built incrementally through *Iterator* and *Observer* pattern application; emphasis on generalization over memorization | Applying patterns across varied contexts using consistent interfaces; generalizing from specific examples | Pattern recognition, selection, and transfer across problem domains | AI can select and apply patterns on demand, eliminating the repeated practice through which recognition develops |
| Eigler et al. (2023) | Systematic review of 12 educational tools; most emphasize detection (83%), application (83%), and selection (67%) | Active problem-solving with immediate feedback; iterative refinement | Pattern detection, application, and selection skills | AI performs all three functions, reducing students to passive evaluators of generated solutions |
| **Contemporary Innovations** | | | | |
| Jeremic et al. (2009) | DEPTHS: collaborative environment combining peer review, software modeling, and implementation | Peer critique, design proposal, iterative implementation, structured feedback cycles | Critical evaluation of design decisions; confidence through social validation | AI replaces the implementation work and potentially the evaluation process these cycles depend on |
| Da Cruz Silva et al. (2019) | Video game as real-world exemplar for pattern application; traditional lecture | Designing game architecture using learned patterns; applying abstract | Concretization of abstract pattern concepts through situated application | AI generates architecturally sound code without students |

| Scholar(s) | Approach | Required Student Activity | Learning Outcome | Vulnerability to AI Substitution |
|---|---|---|---|---|
| | plus game-based assignment | knowledge to a known context | | performing the concretization step |
| Loyola Alvarez et al. (2024) | Serious Video Game with learning and question environments; students choose pattern topics | Active navigation, in-game learning, and answering comprehension questions | Pattern identification (100% of experimental group identified ≥3 patterns); increased motivation | AI can identify and explain patterns, potentially substituting for the active engagement that drove motivation |

## 2.2 The Role of Practice and Struggle

Developing pattern recognition expertise requires more than exposure to pattern definitions—it requires sustained engagement with challenging problems. Research across computer science education and cognitive psychology demonstrates that productive struggle, rather than ease of learning, produces durable expertise. This section examines how expertise develops through deliberate practice and effortful problem-solving, establishing the learning mechanisms that AI code generation tools may disrupt.

Early CS education research recognized this principle implicitly. Wallingford [7] found that novices who learned to work backward from goals to plans through repeated pattern application developed appropriate mental schemas. Proulx [8] engaged students through multiple modalities—lectures, tutorials, assignments, and exams—finding that varied, challenging practice improved exam performance. Jeremic et al. [13] documented that students felt more confident in their learning when engaging in peer evaluation and reviewing multiple solutions to similar problems. These pedagogical approaches, though not explicitly framed as "productive struggle," share a common insight: pattern expertise emerges through active engagement with challenging problems.

### 2.2.1 Software Development Expertise

One of the goals of education, whether in institutions or self-directed, is to develop expertise in the chosen domain. The concept of expertise eludes precise definition, but the presence of expertise is evident in high quality output and performance. The question for us then is how to develop expertise. Popular literature on the topic claims that time alone dedicated to a task will ensure expertise after 10,000 hours [14]. But this is a fallacy. As Ericsson et al.'s [15] research on deliberate practice demonstrated—and subsequent research emphasized in response to this popular misrepresentation [16]—the quality of practice matters far more than mere time-on-task. Expertise develops through effortful, focused engagement with appropriately challenging problems, not through passive accumulation of hours.

Baltes et al. [17] developed a process theory of how software development expertise is cultivated. They interviewed hundreds of software developers with varying levels of experience to determine what practices led to expertise. They did not find a direct relationship between experience and expertise. They discovered that expertise develops through sustained deliberate practice, monitoring, feedback, and self-reflection. This cycle—monitoring performance through feedback and self-reflection, then refining practice—drives increasing expertise.

Software expertise is not only the ability to write code that is better structured and contains fewer bugs, but to have "abstract, transferable knowledge and skills" [17]. This includes knowledge and command of design patterns. Recent research on programming expertise reinforces this finding. Fasco [18] examined how expert software developers differ from novices, finding that experts possess superior pattern recognition skills. In fact, they see fundamentally different patterns than novices see, looking beyond syntax to the deep structures of programs. According to Fasco, expertise represents not merely more efficient information encoding but a fundamental shift toward abstract, transferable knowledge structures. This expertise is only

possible if an individual has metacognitive awareness of one's own capabilities so that practice can be directed for maximum benefit.

### 2.2.2 Practice and Struggle in Computer Science Education

In educational settings, where we are teaching students not only how to program, but also how to learn to program, the role of practice and struggle is paramount. Fasco suggests a three-phase approach to teaching students computer science: *Concept Anchoring*, *Integration*, and *Abstraction* [18]. During the *concept anchoring* phase students are given foundational instruction which they are then required to use during the *integration* phase by applying multiple concepts in complex ways. In the *abstraction* phase students are taught to recognize patterns in various contexts and to identify general problem solving strategies. Unlike traditional instruction, this approach interleaves pattern recognition early in the process alongside syntax instruction.

This structured progression through concept anchoring, integration, and abstraction relies on students encountering challenges at each phase—challenges most visible during debugging. Debugging represents the quintessential site of productive struggle. When novice programmers encounter errors, they must construct mental models of how their code executes, test hypotheses about what went wrong, and revise their understanding when fixes fail [19]. This process—often experienced as frustrating—is precisely where conceptual understanding crystallizes. McCracken et al.'s [20] multi-institutional study revealed that students struggle most with design problems requiring synthesis of multiple concepts, yet this struggle drives expertise development. When implementing design patterns, debugging forces students to understand *why* the pattern's structure matters: an Observer pattern that fails to notify its observers teaches the necessity of proper decoupling more effectively than any lecture [21]. The cognitive work isn't in typing

correct syntax—it's in the iterative process of recognizing failure, diagnosing causes, and reconstructing understanding.

Building on evidence that debugging drives learning, Lowe [22] proposes an even more radical approach: teaching debugging *before* students have command of programming syntax. This deliberately exposes students to concepts beyond their current understanding, requiring them to interact with unfamiliar material—precisely the kind of productive struggle that enhances pattern recognition and mental tracing abilities.

These approaches—whether through structured pedagogical phases (Fasco), error correction (debugging research), or early exposure to unfamiliar code (Lowe)—all leverage productive struggle as the mechanism for expertise development.

### 2.2.3 Desirable Difficulties and Productive Struggle

Research in cognitive psychology reveals why challenging tasks promote expertise: struggle itself serves as a learning mechanism. Two frameworks have demonstrated that appropriate difficulty enhances rather than impedes learning: Bjork and Bjork's [23] [24] "desirable difficulties" and Kapur's [25] [26] "productive failure." Both challenge the assumption that effective instruction should minimize student struggle.

Bjork and Bjork's concept of desirable difficulties identifies specific learning conditions that feel harder but produce more durable understanding. These desirable difficulties include spacing practice over time, interleaving different problem types, and testing before teaching. The mechanism appears to be that difficulty forces deeper processing: when retrieval is easy, learning is shallow; when retrieval requires effort, understanding becomes more robust. Research on desirable difficulties has documented improved retention across domains from vocabulary learning to motor skills [23] [27].

Kapur's theory of productive failure suggests that tasking students with problems before they have received formal instruction increases their learning and retention [25]. He found that students who worked on ill-structured problems engaged with them in complex ways leading to better performance on well-structured problems during a testing phase. Productive failure is a two-stage teaching process [26]. First, students are presented with problems to solve that they do not have the knowledge to solve. Second, they go through formal instruction to consolidate what they have learned. This leads to better outcomes.

While programming education research has not explicitly adopted desirable difficulties or productive failure frameworks, these principles manifest in established pedagogical practices. Students learn debugging through struggle with errors [19], develop pattern recognition through repeated application across varied contexts, and build deeper understanding when forced to solve problems before receiving solutions. The absence of explicit theoretical framing in CS education literature represents a gap this thesis addresses: understanding AI's impact on learning requires first understanding the cognitive mechanisms—productive struggle, effortful problem-solving, iterative refinement—through which programming expertise develops.

**Table 2.2: Mechanisms of Expertise Development Through Practice and Struggle**

| Scholar(s) | Framework/Approach | Required Student Activity | Learning Outcome | Vulnerability to AI Substitution |
|---|---|---|---|---|
| **Software Development Expertise** | | | | |
| Ericsson et al. (1993, 2019) | Deliberate practice: expertise requires effortful, focused engagement with appropriately challenging problems, not mere time-on-task | Sustained, monitored engagement with problems at the edge of current capability | Durable expertise; abstract, transferable knowledge and skills | AI reduces effortful engagement by generating solutions, removing the condition under which deliberate practice occurs |
| Baltes et al. (2018) | Process theory of software expertise: expertise develops through deliberate practice, monitoring, feedback, and self- | Iterative cycle of monitoring performance, receiving feedback, reflecting, and refining practice | Abstract, transferable knowledge including command of design patterns; expertise not | AI short-circuits the feedback-reflection cycle by supplying correct outputs, eliminating the performance monitoring that drives improvement |

| Scholar(s) | Framework/Approach | Required Student Activity | Learning Outcome | Vulnerability to AI Substitution |
|---|---|---|---|---|
| | reflection — not experience alone | | reducible to experience | |
| Fasco (2025) | Experts perceive deep structural patterns invisible to novices; expertise requires metacognitive awareness to direct practice effectively | Self-directed, metacognitively aware practice; recognizing one's own capability gaps to target effort | Fundamental cognitive shift toward abstract, transferable knowledge structures; superior pattern recognition | AI externalizes pattern recognition, preventing students from developing the internal perceptual shift that defines expertise |
| **Practice and Struggle in Computer Science Education** | | | | |
| Fasco (2025) | Three-phase pedagogy: *Concept Anchoring* → *Integration* → *Abstraction*; pattern recognition interleaved early alongside syntax instruction | Applying foundational concepts in complex multi-concept tasks; recognizing patterns across varied contexts | Progressive development from syntax-level understanding to abstract, transferable problem-solving strategies | AI can perform all three phases on behalf of students, collapsing the developmental progression into a single interaction |
| Fitzgerald et al. (2008); Loksa et al. (2016) | Debugging as productive struggle: students construct mental models, test hypotheses, and revise understanding through iterative error correction | Active hypothesis generation and testing; mental tracing of code execution; iterative diagnosis and repair | Conceptual understanding crystallized through failure; internalization of *why* a pattern's structure is necessary | AI resolves errors without requiring students to construct or test mental models, eliminating the mechanism through which understanding crystallizes |
| McCracken et al. (2001) | Multi-institutional study: students struggle most with design synthesis problems requiring integration of multiple concepts | Working through design problems that require synthesizing knowledge across multiple concepts simultaneously | Development of expertise through effortful synthesis; integration of conceptual knowledge into applied capability | AI generates synthesized design solutions, bypassing the integrative cognitive work that drives expertise development |
| Lowe (2019) | Debugging-first pedagogy: students engage with unfamiliar code and concepts before acquiring command of syntax | Interacting with and interpreting code beyond current understanding; productive confusion with unfamiliar material | Enhanced pattern recognition and mental tracing ability; comfort with uncertainty and unknown code | AI resolves unfamiliarity on demand, preventing students from developing tolerance for and skill with novel, uncertain material |
| **Desirable Difficulties and Productive Struggle** | | | | |

| Scholar(s) | Framework/Approach | Required Student Activity | Learning Outcome | Vulnerability to AI Substitution |
|---|---|---|---|---|
| Bjork & Bjork (1994, 2011) | Desirable difficulties: spacing practice, interleaving problem types, and testing before teaching feel harder but produce more durable understanding | Retrieving knowledge under difficulty; working across interleaved problem types; being tested prior to instruction | Robust, durable understanding through deep processing; retention across domains from vocabulary to motor skills | AI converts difficult retrieval into easy lookup, removing the processing depth that desirable difficulties are designed to induce |
| Kapur (2008, 2016) | Productive failure: students work on problems they lack the knowledge to solve before receiving formal instruction | Engaging with ill-structured problems without adequate prior knowledge; generating failed but exploratory solutions | Superior performance on well-structured problems after instruction; deeper retention through prior exploratory failure | AI solves ill-structured problems before students can productively fail, eliminating the exploratory phase that primes subsequent instruction |

## 2.3 AI Tools in Programming Education

Students have long used study aids outside of typical course materials—textbooks, tutoring, online forums—but generative AI represents a fundamentally different kind of tool. Unlike calculators that automate arithmetic or debuggers that locate errors, AI code generators automate the cognitive problem-solving process itself. GenAI can explain concepts, debug code, generate complete solutions from natural language prompts, and provide iterative feedback—combining search engine, tutor, and code generator into a single interface. For programming education, where Section 2.2 established that expertise develops through productive struggle with problems, this poses a unique challenge: AI doesn't just assist with problem-solving; it can replace it entirely.

Research on AI coding tools in education remains nascent, with most studies published in 2024-2025 as tools like ChatGPT and GitHub Copilot gained widespread adoption. The field faces methodological challenges: students use tools in varied ways, different studies examine different AI systems, and pedagogical contexts differ significantly across institutions.

Unsurprisingly, findings are sometimes contradictory—students report both increased productivity and decreased understanding, both confidence in AI suggestions and anxiety about over-reliance. Despite these inconsistencies, several patterns emerge with sufficient consistency to warrant concern. The following sections examine how students use AI tools, how these tools change learning behaviors, and what effects they have on educational outcomes.

Usage of AI coding tools accelerated rapidly among programming students, climbing from 34% in early 2024 to 65-80% by 2025 [28], [29] [30]. Students integrate these tools directly into their development workflows, using them to generate code, debug errors, and explain concepts. While concerns about accuracy persist—with perceptions varying between tools like ChatGPT and GitHub Copilot—these reservations have not prevented widespread adoption. Understanding how students use AI tools requires examining both what they report about the experience and what happens to their learning outcomes.

### 2.3.1 Student Perceptions and Performance

Students report significant productivity gains when using GenAI. Some complete programming assignments 4-5 times faster than manual coding, with higher pass rates on automated tests [31]. Others report feeling more productive, struggling less with syntax, and having mental capacity for higher-order problem-solving [32].

Nearly every study surveyed here contained reports from students that they feared using GenAI would affect either their critical thinking skills, their understanding of their code, or their learning by becoming over-reliant on it [28], [29], [31], [33], [34]. In one study, however, no students reported that using GenAI negatively impacted their skill acquisition [30]. A troubling disconnect emerges between students' expressed anxieties and their behaviors: nearly every study reports students fearing that GenAI will undermine their critical thinking, understanding,

or learning through over-reliance [28], [29], [31], [33], [34], yet these concerns don't translate into reduced usage. Students simultaneously report concerns about dependency while increasing their reliance on these tools.

Research on learning outcomes reveals concerning patterns. Lepp et al. found a statistically significant negative correlation between GenAI use and academic performance [30]. While causality remains ambiguous—weaker students may seek more AI assistance, or AI use may impede learning—the correlation aligns with observed usage patterns: stronger students use AI selectively and critically, while struggling students develop heavier reliance [31]. Most troubling, some students report decreased interest in learning fundamentals and diminished coding confidence after extended AI use, suggesting dependency becomes self-reinforcing [33].

### 2.3.2 Documented Effects on Learning Process

Initial research suggested AI tools might reduce extraneous cognitive load—mental effort spent on syntax errors—while preserving germane load required for understanding [31]. Students report feeling more productive and able to focus on higher-order problems [32]. However, emerging evidence reveals a more troubling pattern: AI tools don't just reduce extraneous load; they allow students to bypass germane cognitive load entirely.

**The cognitive offloading effect.** Rojas-Galeano [35] observed students replacing the problem-solving process itself with prompting. While students reported that GenAI helped them understand syntax and structure, they performed poorly when AI supports were removed during testing. Instructors noted students used GenAI far more extensively than they self-reported, suggesting limited awareness of their own dependency.

**Workflow transformation.** Without GenAI, students follow: *read* → *understand* → *implement*—a sequence that cannot skip the understanding step. With GenAI, this becomes

*prompt → view response → implement*, and eventually simply *prompt → implement* [31]. Students bypass code evaluation entirely, eliminating the step where understanding develops.

**Behavioral patterns.** Prather et al. [34] identified two distinct behaviors: *shepherding* (repeatedly prompting until code works) and *drifting* (evaluating and modifying generated code). Critically, these were student-dependent, not task-dependent—students who shepherded always shepherded, depriving themselves of opportunities to cultivate understanding through evaluation and modification. By bypassing code evaluation, students eliminated the very cognitive work through which programming expertise develops. The workflow transformation isn't merely more efficient—it's categorically different, replacing learning activity with tool interaction.

### 2.3.3 The Theoretical Gap

While these studies document concerning patterns—students bypassing problem-solving, replacing understanding with prompting, and showing performance declines—they lack theoretical frameworks for explaining WHY automation undermines learning. The research describes changed behaviors but cannot analyze what students lose when AI automates the cognitive labor that produces expertise. The next section introduces alienation theory as a framework for understanding how AI restructures students' relationships to their intellectual work.

**Table 2.3: Documented Effects of AI Tool Use on Student Learning Behaviors and Outcomes**

| Scholar(s) | Finding | Evidence/ Method | Implication for Learning |
|---|---|---|---|
| **Student Perceptions and Performance** | | | |
| Baek et al. (2024); Clarke et al. (2025); Lepp et al. (2025) | Rapid adoption: AI tool use among programming students climbed from 34% to 65–80% between early 2024 and 2025 | Survey data across multiple institutions | AI assistance has become a baseline condition of the contemporary programming classroom, not an exceptional case |
| Shihab et al. (2025); | Students report completing assignments 4–5× faster, with higher pass rates and greater mental capacity for higher-order tasks | Self-report surveys; | Perceived productivity gains create incentives for continued and deepened AI |

| Scholar(s) | Finding | Evidence/ Method | Implication for Learning |
|---|---|---|---|
| Takerngsaksiri et al. (2024) | | automated test results | reliance regardless of learning effects |
| Baek et al. (2024); Shihab et al. (2025); Clarke et al. (2025); Nizamudeen et al. (2024); Prather et al. (2024) | Nearly universal student anxiety about over-reliance on AI undermining critical thinking, understanding, and skill acquisition — yet usage continues to increase | Self-report surveys | Students are aware of the dependency risk but lack behavioral strategies or motivation to limit use; concern does not translate to restraint |
| Lepp et al. (2025); Shihab et al. (2025); Nizamudeen et al. (2024) | Statistically significant negative correlation between AI use and academic performance; stronger students use AI selectively while struggling students develop heavier reliance; some students report decreased interest in fundamentals and diminished coding confidence | Longitudinal performance data; usage pattern analysis | AI use may be self-reinforcing: dependency grows as confidence and foundational understanding erode, creating a cycle that compounds over time |
| **Documented Effects on Learning Process** | | | |
| Rojas-Galeano (2025) | Students replace problem-solving with prompting; perform poorly when AI is removed; significantly underreport their own AI usage | Direct observation of student workflows; pre/post testing | AI use degrades transferable skill while creating an illusion of competence; students lose awareness of their own dependency |
| Shihab et al. (2025) | Workflow transformation: *read → understand → implement* becomes *prompt → view → implement* and eventually *prompt → implement*, eliminating the understanding step entirely | Workflow observation and self-report | The cognitive sequence through which understanding develops is structurally bypassed, not merely accelerated |
| Prather et al. (2024) | Two stable behavioral patterns identified: *shepherding* (repeated prompting until code works) and *drifting* (evaluating and modifying generated code); pattern is student-dependent, not task-dependent | Behavioral observation across tasks | Students who shepherd consistently deprive themselves of the evaluative cognitive work that builds expertise; the behavior is habitual, not situational |

## 2.4 Alienation Theory in Learning Contexts

Marx first conceptualized the phenomenon of alienation as something that arises in a mechanized industrial workplace—a result of a worker being a functional cog in an assembly line. The concept of alienation proved useful for understanding the human experience in settings outside of 19th century steam powered factories. Scholars have explored the experience of students in educational institutions through the lens of alienation, mapping the cognitive labor

that a student performs during their education, to the physical labor performed by industrial workers. Though the inputs and outputs differ between industrial labor and modern education, the process is effectively the same. Students take the raw materials of facts, experience, and instruction to produce understanding and transferable skills and knowledge. This body of scholarship establishes two crucial precedents for the present study: first, that alienation theory provides a rigorous framework for analyzing educational processes; and second, that technological mediation of labor creates distinctive forms of alienation—though these two conversations have yet to intersect in analyses of educational technology.

### 2.4.1 Foundational Marxist Analysis of Educational Alienation

Gereluk's foundational 1974 paper argued that alienation in education should not be seen only as a specific psychological state, but rather as a relationship between a student and the institutional power structures in which they do their work [36]. He outlines and criticizes other scholars who in the preceding years attempted to apply Marx's concept of alienation to the student experience. These scholars took an approach that Gereluk describes as "woolly" as they make use of hard-to-define yet incendiary language with terms such as *dehumanization*, *estrangement*, *emasculation*, and *impoverishment*. He critiques the previous approaches for their tautological definition of student alienation that places the cause of alienation within the students themselves. For example, a student is passive which leads to alienation, but a signifier of alienation within a student is passivity. His key insight was methodological: the application of the materialistic dialectic to the educational experience.

Alienation arises when the reasons that a student sought an education are negated by the institution itself. Students are not active participants or drivers in their education, but rather perform tasks that serve the needs of the educators. Social phenomena such as alienation can

only be understood in their social-historical context, and the meaning can be grasped as a union of opposites. In this case a student conceives of themself as an individual in relationship to the people and organizations in their environment. The opposite factors of a student's desire for learning and the demands of the institution, become the conflict which engenders the individual alienation. The tasks that a student performs are mechanized and formulaic, necessarily homogenized, but counterproductive to the development of understanding. Competition among students for rank, favor, and status cause the students to see their peers with mistrust.

Critically, these features are not an accidental byproduct of schools, but rather the purpose. Educational institutions do not place the student as individual at the center of their pedagogical design as schools exist to serve the needs of a capitalist social structure. Gereluk's methodological application of the dialectic to education establishes a basis for future investigations of alienation within education contexts.

### 2.4.2 Alienation as Pedagogical Necessity

In an interesting counterpoint to much extant scholarship on alienation, Kenklies makes the argument that certain forms of alienation are not only unavoidable but pedagogically necessary to create genuine learning [37]. Building upon the work of philosophers such as Plato, Humboldt, and Hegel, Kenklies asserts that alienation is the *conditio sine qua non* for the development of knowledge. Without alienation, an individual consciousness may never know that there is a quality which it lacks and for which it must work to acquire. The theory of *Bildung* put forth by Humboldt posits that in order to achieve intellectual growth, a learner must "reach beyond himself to external objects" and risk "losing himself in this alienation." In the same vein, Hegel's phenomenology sees alienation—self-estrangement and externalization—as "the driving force upon which the development of consciousness towards self-understanding rests."

Kenklies makes a distinction between educational alienation and the classic Marxist form of alienation which views alienation as purely negative. For the purposes of this argument, alienation is a process by which a person becomes disoriented with familiar frameworks, estranged from their current understanding, and separated from the worldview they believe is certain. This has the potential to create a desire to learn something new and acquire new skills. In the realm of education, educators should expose students to concepts and information beyond their current understanding to create this sense of alienation and provoke this intellectual "crisis" which will generate real learning. Nevertheless, such alienation does not always cause transformation and may in fact have the opposite effect of causing a learner to retreat from the uncertainty to the comfort of their previous knowledge.

This *intentional pedagogical alienation* is designed by educators to facilitate growth. Kenklies encourages teachers to create assignments, tasks, and exams with the explicit purpose of destabilizing a student's current understanding. It is markedly different than alienation caused by the learner's relationship to the institutions and environments in which they are learning—the alienation that prevents students from engaging the destabilizing intellectual crisis.

### 2.4.3 Contemporary Applications: Instrumental Learning and Neoliberal Education

Recent scholarship continues to explore the presence of alienation in higher education. Wong provides empirical evidence for student alienation at a community college in Hong Kong [38]. The students at this college approach their education with instrumental rationality. Rather than choose courses in which they have a genuine interest, students choose based on the transferability of the course to four year institutions or the leniency of the instructor. In the courses they do take, they optimize for exam and assignment performance instead of knowledge acquisition or intellectual development. The students approach their time in college from the

perspective of a consumer for the purpose of obtaining a future career. Wong's analysis reveals that students strategize about their courses achieving only the surface understanding necessary to do well in the current course. Instead of deep engagement, students focus on rote memorization, meeting assessment requirements, and managing effort to maximize grades while minimizing intellectual investment. This instrumentalism represents alienation from both the content and process of study—students complete educational labor without genuinely participating in the intellectual work that constitutes authentic learning.

Tan and Li extend this analysis through qualitative research on Chinese college students, identifying three dimensions of learning alienation: alienation from learning processes (loss of ownership as teachers and institutions control educational activities), alienation from "possible selves" (inability to envision meaningful futures through education), and alienation from life rhythms (institutionalized time structures that displace student agency) [39]. Their empirical findings reveal students experiencing learning outcomes as "not belonging to me"—produced to satisfy external requirements rather than to advance their own understanding. When students reported that "learning by oneself" had become their primary mode of education due to inadequate teaching, they expressed not autonomy but abandonment—separation from the collaborative relationship with teachers that should mediate learning.

Ford demonstrates that Marxist educational theory remains a vibrant scholarly tradition, though one still grappling with the relationship between educational critique and revolutionary practice [40]. His analysis, while focused more on political education than on pedagogical alienation specifically, confirms that contemporary scholars continue engaging Marx's framework to examine how educational processes either reproduce or challenge existing social relations.

### 2.4.4 Technology-Mediated Labor Alienation

While educational scholars have examined alienation in learning contexts, a parallel body of scholarship has investigated how technological innovation creates alienation in workplace settings. Karayaman demonstrates how automation in contemporary organizations generates powerlessness as technology displaces worker control over production processes [41]. Employees experience their work as increasingly meaningless as they become "part of a system that operates like a machine"—what Karayaman terms "digital Taylorism." This technological mediation creates multiple dimensions of alienation: workers feel they lack the qualifications required by new technologies, experience their labor as routine and repetitive rather than creative, and find themselves excluded from decision-making processes now controlled by algorithmic systems.

However, these analyses of technology-induced alienation focus exclusively on workplace labor—manufacturing, service work, platform employment—rather than on educational contexts. The technological disruption they examine concerns workers producing commodities for employers, not learners producing understanding for themselves. The separation of workers from control over production processes shares structural similarities with educational alienation, but the contexts, purposes, and relations of production differ fundamentally.

**Table 2.4: Alienation Theory: Frameworks and Applications Relevant to AI-Assisted Learning**

| Scholar(s) | Framework/Concept | Form of Alienation Identified | Mechanism | Application to AI-Assisted Programming Education |
|---|---|---|---|---|
| **Foundational Marxist Analysis of Educational Alienation** | | | | |
| Gereluk (1974) | Materialist dialectic applied to education; alienation as structural relationship between student and institutional power, | Alienation arising from the conflict between students' desire for learning and the mechanized, homogenized | Students perform formulaic tasks that serve institutional needs rather than their own | Establishes the methodological basis for analyzing AI-assisted education structurally — AI may extend institutional |

| Scholar(s) | Framework/Concept | Form of Alienation Identified | Mechanism | Application to AI-Assisted Programming Education |
|---|---|---|---|---|
| | not a psychological state internal to the student | demands of educational institutions | development; competition among peers produces mistrust | mechanization into the cognitive labor students perform |
| **Alienation as Pedagogical Necessity** | | | | |
| Kenklies (2022) | *Intentional pedagogical alienation*: drawing on Humboldt and Hegel, argues that certain forms of alienation are the *conditio sine qua non* of genuine learning | Productive alienation: disorientation from familiar frameworks that generates the intellectual crisis necessary for growth | Educators deliberately expose students to concepts beyond current understanding to provoke crisis and motivate knowledge acquisition | Distinguishes productive alienation (designed to facilitate growth) from alienation that forecloses engagement — AI may eliminate the former while producing the latter |
| **Contemporary Applications: Instrumental Learning and Neoliberal Education** | | | | |
| Wong (2022) | Instrumental rationality in higher education; students approach learning as consumers optimizing for credentials, not knowledge | Alienation from learning content and process: surface understanding sufficient for assessment, without genuine intellectual participation | Students strategize around exams and assignments, achieving rote memorization and effort minimization rather than deep engagement | AI enables and accelerates instrumental rationality — students can meet assessment requirements with even less intellectual investment |
| Tan & Li (2025) | Three-dimensional model of learning alienation in contemporary higher education | (1) Alienation from learning processes (loss of ownership); (2) alienation from "possible selves" (inability to envision meaningful futures through education); (3) alienation from life rhythms (institutionalized time displacing student agency) | Institutional structures produce learning outcomes experienced as "not belonging to me" — external requirements replace self-directed understanding | AI may intensify all three dimensions: students lose ownership of the problem-solving process, disconnect capability from identity, and have their learning rhythms restructured around tool interaction |
| **Technology-Mediated Labor Alienation** | | | | |
| Karayaman (2024) | Digital Taylorism: automation in | Alienation through automation: workers | Technology mediates labor to | Provides the closest structural analogue to |

| Scholar(s) | Framework/Concept | Form of Alienation Identified | Mechanism | Application to AI-Assisted Programming Education |
|---|---|---|---|---|
| | contemporary organizations generates powerlessness as technology displaces worker control over production | feel unqualified for technologized roles, experience labor as routine rather than creative, and are excluded from algorithmic decision-making | the point where workers become functional components of a machine system rather than agents of production | AI-assisted programming education: cognitive labor displaced by automation, with the learner reduced from agent to monitor |

### 2.4.5 The Gap: Educational Technology and Cognitive Labor

This review reveals two established scholarly conversations that have not yet intersected. Educational alienation scholars examine how pedagogical design [37], institutional structures [38], [39], and political-economic contexts [40] affect students' relationship to learning—but they do not examine technological mediation of educational processes. Technology-alienation scholars examine how automation disrupts workers' control over labor processes [41]—but they analyze workplace production, not educational cognitive labor.

The two scholarly conversations reviewed in this section have not intersected around the question of how AI automation creates alienation in the cognitive labor through which expertise develops. This gap is particularly significant for programming education, where students must develop "pattern thinking" through sustained engagement with design problems. When AI tools automate the cognitive labor of translating requirements into code, recognizing appropriate design patterns, and synthesizing solutions, they potentially create new forms of pedagogical alienation. Unlike Kenklies's intentional alienation designed to facilitate growth, and unlike Karayaman's workplace automation that displaces physical or routine cognitive labor, AI-assisted programming education may automate precisely the intellectual work through which expertise emerges. Students may produce working code without participating in the problem-

solving processes that generate genuine understanding—a form of alienation that existing scholarship has not addressed.

# 3 Related Work

Research on AI coding tools has documented troubling patterns: students bypass problem-solving, show performance declines on independent assessments, and develop dependency behaviors. Yet these empirical studies lack theoretical frameworks to explain why these patterns matter or how AI fundamentally transforms learning. Meanwhile, critical pedagogues have long warned about educational practices that alienate students from intellectual work—but they wrote before AI existed in classrooms. This chapter examines both bodies of scholarship to establish a crucial gap: no research has systematically analyzed how AI automation of cognitive labor creates pedagogical alienation in programming education. Section 3.1 shows what empirical studies document but cannot explain. Section 3.2 demonstrates how critical pedagogy provides explanatory frameworks these studies lack. Section 3.3 synthesizes these literatures to reveal the need for theoretically-grounded empirical investigation of AI's impact on the cognitive labor through which programming expertise develops.

## 3.1 Empirical Evidence of Problems with AI Use in Programming Pedagogy

In order to show how AI disrupts the development of transferable knowledge in programming education, we must understand the cognitive processes that drive the development of this expertise. We previously discussed two complementary ideas that have been shown to enhance learning in students: *desirable difficulties* and *productive failure* [24], [25], [27]. Both frameworks demonstrate that effortful cognitive work—not ease of learning—produces durable expertise. Difficulty isn't a pedagogical flaw to be eliminated; it's the mechanism through which understanding becomes robust—it's a feature, not a bug.

**Table 3.1: Three Frameworks for Expertise Development Through Effortful Engagement: Similarities, Differences, and Points of AI Disruption**

| **Dimension** | **Desirable Difficulties** (Bjork & Bjork, 1994; 2011; Schmidt, 1992) | **Productive Failure** (Kapur, 2008; 2016) | **Deliberate Practice** (Ericsson et al., 1993; Baltes et al., 2018) |
|---|---|---|---|
| **Core claim** | Conditions that feel harder in the short term produce more durable long-term retention | Struggling with problems *before* instruction produces better outcomes than receiving instruction first | Expertise develops through focused, effortful engagement with appropriately challenging problems — not through time-on-task alone |
| **Theoretical origin** | Cognitive psychology; memory and retention research | Educational psychology; mathematics education | Expertise studies; performance science |
| **Primary mechanism** | Effortful retrieval strengthens memory traces and forces elaborative encoding | Prior exploration activates and differentiates knowledge structures that formal instruction can then consolidate | Iterative cycles of practice, monitoring, feedback, and self-reflection build increasingly refined mental representations |
| **Role of failure** | Implicit — difficulty implies unsuccessful retrieval attempts, but failure is not the explicit focus | Explicit — unsuccessful attempts *are* the productive phase; failure is the pedagogical instrument | Reframed as diagnostic — failures reveal gaps that direct subsequent practice toward maximum benefit |
| **Timing of difficulty** | Distributed throughout learning via spacing, interleaving, and testing | Front-loaded *before* formal instruction | Continuous and self-directed; difficulty is deliberately introduced at each stage of development |
| **Role of instruction** | Precedes difficulty; desirable difficulties arise during retrieval and application | Follows difficulty; instruction consolidates what exploration revealed | Serves as a source of feedback and calibration; the learner directs their own engagement |
| **Learner awareness required** | Low — desirable difficulties operate even when learners are unaware of them | Partial — learners must engage with the problem rather than disengage, but need not understand why failure is productive | High — metacognitive awareness of one's own capability gaps is essential for directing practice effectively |
| **Domain of original evidence** | Motor skills, vocabulary, factual recall — broad across domains | Mathematics and physics — structured problem domains | Music, chess, sports — expert performance domains |
| **Application to programming education** | Spacing practice across problem types; interleaving syntax and design tasks; testing before teaching new patterns | Assigning design problems before teaching relevant patterns; debugging unfamiliar code before formal instruction (cf. Lowe, 2019) | Expert developers self-impose challenges; students need guided exposure to increasingly complex problems with structured feedback (cf. Fasco, 2025) |
| **Point of AI disruption** | AI converts effortful retrieval into frictionless lookup, removing the processing depth that produces retention | AI solves the problem before students can productively fail, eliminating the exploratory phase that primes instruction | AI substitutes for the feedback-reflection cycle, preventing the self-monitoring through which practice becomes deliberate |
| **Key convergence** | All three frameworks identify effortful cognitive engagement — not ease of performance — as the mechanism | | |

| **Dimension** | **Desirable Difficulties** (Bjork & Bjork, 1994; 2011; Schmidt, 1992) | **Productive Failure** (Kapur, 2008; 2016) | **Deliberate Practice** (Ericsson et al., 1993; Baltes et al., 2018) |
|---|---|---|---|
| | through which durable expertise develops | | |
| **Key divergence** | Difficulty is distributed and ongoing; the unit of analysis is the retrieval event | Difficulty is concentrated before instruction; the unit of analysis is the problem-solving episode | Difficulty is continuous and self-regulated; the unit of analysis is the practice career |

Productive failure and desirable difficulties may seem counterintuitive at first: how does struggling to complete a task produce more learning than successfully completing it? When a student does not know how to solve a problem immediately, they spend a greater amount of time on the problem often learning more ways that don't work than ways that do work. This process adds more nodes to the knowledge network—each node representing a failure is a trail marker on the path to the solution. Through deliberate repeated activation of the knowledge network during the problem solving process—as new nodes are added and previously traced paths are traversed again—the individual gains a greater ability to activate the knowledge network in the future. Additionally, experts have superior pattern recognition skills. To identify a pattern, one must see all the parts of the problem space that are not the pattern. Only through repeated explorations in the problem space can the skill of recognition be developed.

For programming education specifically, these cognitive mechanisms manifest in common tasks that both students and professionals perform. Fitzgerald et al. [19] and Lowe [22] showed that debugging increased the learning of students in programming classes. Debugging is a form of *effortful retrieval* which Bjork [23] identified as a "potent learning event." During debugging a developer creates a mental representation of the code and maps that to the knowledge they have of the programming language, the design patterns they know, the programming principles they have learned. Discovering why a program is not working as intended can bring into union multiple concepts.

To develop pattern recognition skills, students need exposure to patterns in wide-ranging contexts [20], [21]. Kapur's productive failure gives students opportunity to search for patterns in service of designing a solution to problems for which they have incomplete information [25]. Pattern thinking is seeing the negative space around the pattern as well as seeing the pattern itself. Productive failure helps learners to see what parts of the problem do not match the pattern. Through repeated exploration in the problem space, the skill of pattern detection is increased and a learner develops an intuition about where to apply patterns and where to look to identify them.

Baltes [17] surveyed expert developers who reported that they engage in sustained deliberate practice [17], or in other words they introduce *desirable difficulties* into their learning. Such developers know not only to practice a skill, but also to reflect on their learning and get feedback. This reflection and feedback informs developers of where they should create difficulties for themself during the next phase of learning. The cycle itself—deliberate practice, monitoring, feedback, and self-reflection—instantiates *effortful retrieval* that produces robust understanding.

These frameworks converge on a crucial insight: programming expertise develops through the same effortful cognitive processes that general learning theory identifies as essential. The difference lies not in the mechanism but in the domain—where Bjork studies vocabulary retention, Fasco studies pattern recognition [18]; where Kapur examines math problem-solving, Baltes examines software development. The underlying principle remains constant: expertise emerges from sustained engagement with appropriately challenging problems. However, while these frameworks establish the cognitive mechanisms through which programming expertise develops—effortful problem-solving, productive struggle with challenging tasks, sustained deliberate practice with critical feedback—they cannot address what occurs when technological

mediation eliminates the very cognitive labor they identify as essential. What becomes of learning when the very cognitive labor that produces expertise is automated away?

When AI tools automate problem-solving, they represent a qualitatively different intervention than traditional shortcuts. AI tool use is not akin to other shortcuts in learning that a student may take. It presents a significant change in the way a student works and therefore it opens new pathways to alienation. Previous to the advent of tools like *Github Copilot* students may have found solutions to their program problems online and then copied and modified the code for use in the assignments. This is a *creative* task. A student needs to *understand* the problem they are facing well enough to use a search query that will return relevant results, understand the code they read well enough to *know* that it is a possible solution, and then *alter* the solution so it fits in with their existing code. AI tools eliminate this creative cognitive work—precisely the effortful retrieval and productive struggle that we have established as essential for expertise development.

We just saw that this process is valuable and a necessary part of transformative learning. The students who make use of AI when learning programming sense this as well. Nevertheless, recent research has not yet explored the effects or consequences of AI tools on a learner's learning. Researchers focus on quantifiable metrics such as task performance and correctness [31], [42] and the manner in which an AI tool is used to execute a task [34]. The investigators who explore qualitative aspects of classroom uses of AI look at students' perceptions of their own use [28], [32], [35]. These approaches document symptoms without diagnosing disease—they cannot explain why the patterns they observe matter for learning itself.

The studies that survey students about their individual experience with AI tools to complete programming assignments raise some concerns that cannot be dismissed as part of an

adjustment period while we learn how to incorporate AI into our learning. Some students expressed concern that they could easily become over-reliant on AI coding tools, while others feared that it would hinder their critical thinking skills or that it would do too much of the work allowing them to avoid thinking through the problem [28], [32]. Still others have reported being less confident in their understanding of programming fundamentals or coding skills when they learn in conjunction with an AI [33]. Yet these studies treat student anxiety as subjective perception rather than recognizing it as evidence of objective transformation in learning processes. Without frameworks to analyze how AI restructures cognitive labor, researchers cannot validate or dismiss these concerns—they can only document them.

The concerns these students express are valid. Rojas-Galeano observed a cognitive offloading effect by students whereby they elided the problem-solving process altogether [35]. Without the aid of AI, students were unable to perform similar coding tasks. This indicates that the AI tool use was an impediment, rather than a support, to learning even though the students expressed confidence in their learning. Additionally, students self-reported far less AI use than instructors observed in the lab. Students are aware of the possibility of over-reliance, but are exhibiting a bias blind spot—over-reliance is a concern, but they are not themselves over-reliant. These findings raise questions that the study does not address: Why do students feel confident in their learning when they use AI even though they are not in fact mastering the material? Why are students unaware of the degree to which they use AI?

Prather et al. described the two novel behaviors of *shepherding* and *drifting* that students manifested when using GitHub Copilot for coding tasks. They suggested that shepherding indicates an over-reliance on AI, while drifting indicates that a student is being deliberate and thinking critically, yet their framework cannot explain why students adopt one over the other—

treating behavioral patterns as given rather than as symptoms requiring theoretical analysis [34]. The authors see drifting as the more desirable of the two behaviors, but there is no explanation as to the factors that lead a student to engage in shepherding instead of drifting, nor how we could design curriculum to encourage students to use drifting and be more critical of the code that Copilot produces. Overall, the authors are encouraged by Copilot and think that it could help students with metacognitive scaffolding so that they can avoid the blank page problem and focus on higher-order concerns like program structure or semantics. Their findings indicate otherwise. In fact, Copilot's interaction model can get in the way of learning. Students who had less understanding about how to solve the problem found Copilot's suggestions inscrutable, leading to a temptation to accept the suggestions even though they could not explain how it worked. The high degree of trust that students have in Copilot's code lessens the likelihood of a student critiquing the code that Copilot generates. Students who do not know what to do next can use Copilot to move on without struggling to see the bridge between the solved and unsolved parts of the problem. Yet recognizing these connections—what to do next—is precisely the skill programmers need to develop. Research on AI efficiency reinforces this concern.

GenAI tools for programming promise an efficiency improvement, and all of the studies on its use in programming education show that students are indeed more efficient. Shihab et al. reported students were four to five times more efficient, as well as more accurate, when using Copilot than when not using it [31]. This efficiency may come at a cost. Part of the reason for the efficiency is that Copilot allows students to skip the understanding step in the problem-solving process. They move quickly from an interaction pattern with Copilot of *prompt → view response → implement* to a pattern of *prompt → implement*. The optimism of Prather that Copilot will free students from the cognitive load of syntax and the blank page is shown by Shihab to be

unfounded. Instead of seeing students freed from extraneous cognitive load (syntax) and engaging in intrinsic (the programming problem) and germane (the actual learning) cognitive load, they saw students become cognitively disengaged. High performing students were more likely to be judicious in their AI use while struggling students used it more frequently. This means that if AI does in fact disrupt the learning process, the poor performers will remain poor performers. These concerns are raised, but not addressed systematically. The researchers suggest developing new learning outcomes, such as prompt engineering and metacognitive thinking skills, for GenAI assisted programming. Without theoretical grounding, they cannot explain *why* high performers use AI judiciously while struggling students increase dependency—perpetuating rather than addressing performance gaps. They document the pattern without analyzing the mechanism.

The theme in these studies is that GenAI is a net benefit for the classroom and with some optimizations to our teaching methods we can continue to ensure the success of our students. Even though these researchers present findings indicating cognitive disengagement and interrupted learning processes, they all suggest what amounts to the same thing: adjust curriculum to increase engagement with the problem solving process when students use GenAI. We see that students may be losing something, but we do not yet know what that is. Without a robust theoretical framework we do not have a map for how to build these new curricula, nor do we have a way to evaluate their effectiveness. This thesis will present a theoretical framework through which we can investigate how students experience their own learning and what changes when they learn with the help of GenAI.

**Table 3.2: Publications Informing Section 3.1: Cognitive Mechanisms, CS Education Evidence, and Empirical AI Research**

| Scholar(s) | Domain | Core Finding or Claim | Cognitive Mechanism Identified | Analytical Limitation |
|---|---|---|---|---|
| **Cognitive Frameworks** | | | | |
| Bjork & Bjork (1994; 2011); Schmidt (1992) | Cognitive psychology; memory and retention | Conditions that feel harder in the short term — spacing, interleaving, testing before teaching — produce more durable long-term retention | Effortful retrieval forces deep processing, strengthening memory traces and producing robust understanding | Developed outside educational technology contexts; cannot address what happens when retrieval effort is eliminated by external tools |
| Kapur (2008) | Educational psychology; mathematics education | Students who struggle with problems before receiving formal instruction outperform those who receive instruction first | Prior exploratory failure activates and differentiates knowledge structures that instruction then consolidates | Does not examine technological mediation; assumes students engage with the problem rather than delegating it |
| Ericsson et al. (1993); Baltes et al. (2018) | Expertise studies; software development | Expertise develops through deliberate, effortful practice with monitoring and self-reflection — not through time-on-task alone | Iterative cycles of practice, feedback, and self-reflection build increasingly refined mental representations | Analyzes human practice in isolation; does not account for AI tools that substitute for or short-circuit the feedback-reflection cycle |
| **CS Education Research** | | | | |
| Fitzgerald et al. (2008); Lowe (2019) | Computer science education | Debugging increases student learning; exposing students to unfamiliar code before syntax instruction enhances pattern recognition and mental tracing | Debugging is a form of effortful retrieval that unifies multiple concepts through the process of diagnosis and repair | Documents the learning value of debugging but does not examine how AI-assisted debugging changes or forecloses this mechanism |
| McCracken et al. (2001) | CS education; multi-institutional study | Students struggle most with design problems requiring synthesis of multiple concepts, yet this struggle drives expertise development | Effortful synthesis across concept domains produces integrative understanding that cannot be achieved through exposure alone | Identifies the centrality of synthesis struggle without addressing conditions under which that struggle might be automated away |
| Loksa et al. (2016) | CS education; design pattern learning | Debugging a failing Observer pattern teaches the necessity of | Failure makes abstract design principles concrete by revealing the | Does not examine AI-assisted environments where failures may be |

| Scholar(s) | Domain | Core Finding or Claim | Cognitive Mechanism Identified | Analytical Limitation |
|---|---|---|---|---|
| | | decoupling more effectively than any lecture | consequences of violating them | resolved before students encounter or interpret their consequences |
| Fasco (2025) | CS education; expertise development | Experts perceive deep structural patterns invisible to novices; expertise requires metacognitive awareness to direct practice; proposes three-phase pedagogy: Concept Anchoring → Integration → Abstraction | Expertise represents a fundamental perceptual shift toward abstract, transferable knowledge structures, enabled by metacognitive self-direction | Three-phase model assumes students engage in each phase; does not analyze what happens when AI compresses or eliminates phases |
| **AI Tools Research** | | | | |
| Shihab et al. (2025) | AI tools in programming education | Students using Copilot are 4–5× more efficient and more accurate, but collapse the problem-solving workflow from *prompt → view → implement* to *prompt → implement*, eliminating the understanding step | Identifies workflow transformation and cognitive disengagement in lower-performing students | Documents the pattern without theoretical grounding; cannot explain *why* high performers use AI judiciously while struggling students increase dependency |
| Prather et al. (2024) | AI tools in programming education | Two stable behavioral patterns emerge: *shepherding* (repeated prompting until code works) and *drifting* (evaluating and modifying generated code); pattern is student-dependent, not task-dependent | Identifies behavioral typology of AI-assisted programming but treats patterns as given rather than as symptoms | Cannot explain what produces shepherding versus drifting, nor how curriculum might be designed to encourage more deliberate engagement |
| Rojas-Galeano (2025) | AI tools in programming education | Students replace problem-solving with prompting; perform significantly worse when AI is removed; substantially | Documents cognitive offloading and a bias blind spot: students believe they are learning while evidence indicates otherwise | Raises the questions — why do students feel confident while not mastering material? — but lacks the theoretical framework to answer them |

| Scholar(s) | Domain | Core Finding or Claim | Cognitive Mechanism Identified | Analytical Limitation |
|---|---|---|---|---|
| | | underreport their own AI use | | |
| Takerngsaksiri (2024); Baek et al. (2024); Nizamudeen et al. (2024) | Student perceptions of AI tools | Students report productivity gains alongside widespread anxiety about over-reliance, declining critical thinking, and reduced confidence in fundamentals — yet usage continues to increase | Identifies a disconnect between expressed concern and behavior, but treats student anxiety as subjective perception | Without theoretical frameworks, cannot validate or dismiss these concerns; documents symptoms without diagnosing the underlying transformation |
| Wang et al. (2025) | AI efficiency in programming tasks | Confirms efficiency gains from AI assistance across programming tasks | Quantifies performance improvements on discrete tasks | Focuses on task-level metrics; cannot address effects on the development of transferable expertise over time |

## 3.2 Critical Perspectives

The field of critical pedagogy has long sought to understand how an education system functions in the context of power and social structures in the culture at large. These investigations have led to insights into how students and teachers interact in the classroom and how students engage with their own learning. From the perspective of critical pedagogy, the purpose of education is self and social empowerment and this is *ethically prior* to the acquisition and mastery of technical skills that are based on the logic of the marketplace [43]. Knowledge is viewed as a social construct—not facts themselves, but the selection of which facts matter, which questions get asked, and whose interests these serve. When discussing knowledge here, we do not mean facts are arbitrary—critical pedagogy doesn't argue the sky is blue because different interests agreed on this. Rather, what counts as legitimate knowledge, which facts are worth teaching, and whose questions matter are socially determined. We teach that the sky is blue

because this knowledge serves purposes deemed valuable by those with power to shape curricula.

In his work *Pedagogy of the Oppressed*, Paulo Freire describes an education system that uses what he calls the *banking model* [44]. In the banking model, knowledge is deposited into the student by the instructors. The relationship between student and teacher is unidirectional as the student is only a passive receiver of knowledge whose main task is to store the deposits. This reveals a fundamental blind spot in AI coding tools research. When Prather identifies the shepherding behavior, they document students repeatedly prompting GenAI until they get a solution that works, but their framework cannot explain why this is pedagogically problematic beyond noting that students "miss opportunities" [45]. Freire provides the missing analysis: shepherding is the banking model perfected—students receive deposits of AI-generated solutions without the problem-posing dialog through which genuine understanding develops. This is an automated banking system which eliminates the possibility of what Freire calls "praxis"—the unity of reflection and action. AI automates the action, but the cognitive labor of reflection is skipped altogether when students shepherd their way to a solution.

Freire's insight that "the means used are not important; to alienate human beings from their own decision-making is to change them into objects" proves chillingly prescient in the age of AI assistance [44]. Whether alienation occurs through authoritarian banking-model instruction or through students' voluntary adoption of AI tools matters less than the structural result: in both cases, learners are prevented from "engaging in the process of inquiry" and thereby transformed from subjects into objects. GenAI becomes another means—perhaps more effective precisely because it feels like empowerment rather than oppression.

Michael Apple provides insight into a process called *deskilling* whereby complex tasks requiring a high degree of skill are broken down into constituent parts until these individual tasks can be performed by an unskilled laborer [46]. Once an individual has been deskilled to a certain point, they are then reskilled for another purpose within the same context. For example, a teacher once engaged in the work of designing a curriculum, but with the ability to purchase prepackaged curriculum that can be taught by anyone who has the materials, their creative productive skills are replaced with skills to manage and control a classroom.

Rojas-Galeano saw this exact process in his students, but did not name it as such [35]. The students he taught offloaded the problem-solving process to GenAI, replacing this fundamental skill with those of prompt engineering and snippet assembly. Students are unintentionally deskilling by relying on AI to solve their programming problems. With GenAI, programming can be turned from a creative endeavor to a sequencing of atomistic tasks. This sequencing is what Shihab observed when students using Copilot collapsed programming from three distinct steps into two, eliminating the problem-solving phase entirely. Programming with GenAI no longer requires higher-order problem-solving skills but now "requires only the mastery of the prior necessary technical skills and enough time to follow the rules, at one's own pace, to their conclusion" [46].

Critical pedagogy has begun finding application in computer science education. Mayhew and Patitsas document computing educators applying Freire's banking model critique to CS classrooms, seeking to disrupt power structures and develop students' critical consciousness about technology's societal impacts [47]. However, their focus—like most critical pedagogy in CS—centers on social justice and ethical dimensions of computing rather than on how

educational technology itself mediates the learning process. The question of how AI tools restructure students' relationship to cognitive labor remains unaddressed.

**Table 3.3: General CS Education Pedagogical Frameworks: Theoretical Traditions, Views of Learning, and Limitations for AI Contexts**

| Framework | Scholar(s) | Theoretical Tradition | View of Learning | Student's Role | View of Technology | Limitation for AI Context |
|---|---|---|---|---|---|---|
| Banking Model / Problem-Posing | Freire (2018) | Critical pedagogy | Learning as praxis — the unity of reflection and action | Active subject in problem-posing education; passive object in banking-model education | Technology as a potential instrument of banking-model deposit delivery | Predates AI; describes structural alienation from inquiry but not the automation of cognitive labor itself |
| Deskilling / Reskilling | Apple (2012) | Critical pedagogy; labor theory | Skill development through creative intellectual labor | Creative agent whose skills can be externalized by technological systems | Technology as an instrument of deskilling — separates workers from the labor that constitutes their expertise | Developed in workplace contexts; the structural analysis applies but the educational cognitive labor dimension is unaddressed |
| Educational Alienation | Gereluk (1974) | Marxist materialist dialectic | Learning as self-directed development in dialectical tension with institutional demands | Individual in structural conflict with institutional power | Not addressed | Establishes the methodological basis for analyzing alienation in education but does not examine specific alienating mechanisms or technological mediation |
| Intentional Pedagogical Alienation | Kenklies (2022) | Philosophy of education (Humboldt, Hegel) | Productive disorientation as the precondition for genuine growth | Active learner who must risk losing themselves in unfamiliar conceptual territory | Not addressed | Distinguishes productive from destructive alienation but does not examine how technology might |

| Framework | Scholar(s) | Theoretical Tradition | View of Learning | Student's Role | View of Technology | Limitation for AI Context |
|---|---|---|---|---|---|---|
| | | | | | | eliminate the former while producing the latter |
| Instrumental Rationality | Wong (2022) | Critical sociology | Surface learning oriented toward credentials rather than understanding | Consumer optimizing for grades, transferability, and effort minimization | Not addressed | Describes behavior that AI enables and accelerates but does not analyze AI's role in producing or deepening it |
| Three-Dimensional Learning Alienation | Tan & Li (2025) | Critical sociology; qualitative empiricism | Alienation as loss of ownership across three dimensions: learning processes, possible selves, and life rhythms | Agent whose ownership of learning is progressively expropriated by institutional structures | Not addressed | Rich empirical framework but no mechanism for analyzing how specific technologies mediate or intensify each dimension of alienation |
| Digital Taylorism | Karayaman (2024) | Labor theory; organizational sociology | Automation displaces skilled labor, reducing workers to monitors of algorithmic systems | Worker as functional component of a machine system rather than agent of production | Technology as the primary alienating force; automation generates powerlessness | Analyzes workplace labor; the structural analysis is directly applicable but the context — productive labor, not educational cognitive labor — differs fundamentally |
| Three-Phase CS Pedagogy | Fasco (2025) | CS education; expertise studies | Progressive development through Concept Anchoring, Integration, and Abstraction | Active learner developing metacognitive awareness to direct their own practice | Not addressed | Pedagogical design framework; does not theorize what happens when AI compresses or eliminates the phases |

| Framework | Scholar(s) | Theoretical Tradition | View of Learning | Student's Role | View of Technology | Limitation for AI Context |
|---|---|---|---|---|---|---|
| | | | | | | students must traverse |
| Critical Pedagogy in CS | Mayhew & Patitsas (2023); McLaren (2016) | Critical pedagogy; social justice | Education as a site of social transformation and critical consciousness about technology's role | Student developing critical consciousness about technology's societal impacts | Subject of critical analysis — technology's social and ethical consequences | Focuses on technology as a topic of critical inquiry rather than as a mediator of the learning process itself |

**Table 3.4: Pattern Learning Pedagogical Frameworks: Strategies, Required Student Activity, and Limitations for AI Contexts**

| Framework | Scholar(s) | Pattern Type | Pedagogical Strategy | Required Student Activity | Learning Outcome | Limitation for AI Context |
|---|---|---|---|---|---|---|
| OOP Programming Patterns | Wallingford (1996) | Programming patterns: *View*, *State*, *Decorator* | Introduce simple, concrete patterns to scaffold pre-coding mental schema formation | Apply patterns to complete assignments; work through initial resistance to adopt structured approaches | Pre-coding mental schema; ability to plan program structure before writing | AI generates pattern-conforming code without students constructing the schema the patterns are designed to develop |
| Process Patterns | Proulx (2000) | Ten generalized process patterns (e.g., *Name Use*, *Cumulative Results*) | Scaffold the transition from blank editor to working program through sequenced decision-making | Name components, sequence decisions, combine patterns across multi-step problems | Compositional reasoning; understanding of program structure as a series of deliberate decisions | AI collapses the sequencing process into a single prompt, bypassing each decision point the patterns are designed to make explicit |
| Design Vocabulary Building | Astrachan et al. (1998) | GoF design patterns, beginning with *Iterator* and *Observer* | Build design vocabulary incrementally; emphasize generalization over memorization of pattern specifications | Apply patterns across varied contexts using consistent interfaces; generalize concepts beyond formal pattern definitions | Pattern recognition, selection, and transfer across problem domains and languages | AI selects and applies patterns on demand, eliminating the repeated cross-context practice through which recognition and transfer develop |

| Framework | Scholar(s) | Pattern Type | Pedagogical Strategy | Required Student Activity | Learning Outcome | Limitation for AI Context |
|---|---|---|---|---|---|---|
| GoF Design Pattern Taxonomy | Gamma et al. (1994; 2011) | 23 canonical design patterns defined by four-part schema: name, problem, solution, consequences | Shared vocabulary as the foundation for design discussion and reuse | Recognize problem contexts, select appropriate patterns, apply and evaluate trade-offs | Reusable solution templates and a shared professional design language | Expert-level reference framework; provides taxonomy but not pedagogy; does not address what happens when AI performs pattern selection and application automatically |
| Tool-Based Pattern Education | Eigler et al. (2023) | Design patterns across 12 reviewed educational tools | Active learning tools emphasizing pattern detection (83% of tools), application (83%), and selection (67%) | Interactive problem-solving with immediate feedback; iterative refinement across detection, application, and selection tasks | Integrated pattern detection, application, and selection skills | AI performs all three functions on demand, reducing students to passive evaluators of generated solutions rather than active pattern thinkers |
| DEPTHS | Jeremic et al. (2009) | Design patterns | Collaborative environment integrating peer review, software modeling, implementation, and structured feedback cycles | Critique peer proposals, design solutions, implement, present to peers, and evaluate multiple solutions | Critical evaluation of design decisions; confidence and understanding developed through social validation and iterative feedback | AI can substitute for the implementation work around which the collaborative feedback cycles are structured, hollowing out the activity the social learning depends on |
| Game-Based Pattern Learning | Da Cruz Silva et al. (2019); Loyola Alvarez et al. (2024) | Design patterns | Video games as motivating learning environment or real-world exemplar for applying patterns in | Navigate game environment, apply learned patterns, design game architecture | Concretization of abstract pattern concepts through situated application; increased | AI generates architecturally sound, pattern-conforming solutions without students |

| Framework | Scholar(s) | Pattern Type | Pedagogical Strategy | Required Student Activity | Learning Outcome | Limitation for AI Context |
|---|---|---|---|---|---|---|
| | | | situated contexts | using known patterns | motivation and identification of ≥3 patterns | performing the concretizatio n step the game context is designed to elicit |
| Three-Phase Pattern Pedagogy | Fasco (2025) | Pattern recognition as the central learning outcome | Concept Anchoring → Integration → Abstraction; pattern recognition interleaved with syntax instruction from the outset | Engage effortfully at each phase; develop metacognitiv e awareness of capability gaps to direct practice | Progressive development from syntax-level understandin g to abstract, transferable pattern recognition and deployment | AI compresses or eliminates developmental phases; students may reach surface-level performance without traversing the stages through which genuine pattern recognition develops |

## 3.3 The Unimplemented Method

The literature reveals a troubling disconnect. Empirical studies document concerning patterns—students bypassing problem-solving, performance declining on independent assessments, dependency behaviors emerging—yet lack theoretical frameworks to explain why these patterns matter beyond efficiency metrics. Researchers note students 'miss opportunities' for learning but cannot analyze what cognitive labor is being automated away or how this transforms the relationship between learner and intellectual work.

Critical pedagogy provides the missing analytical tools. Freire's banking model reveals how AI perfects deposit-based education, eliminating the problem-posing inquiry essential for authentic learning. Apple's deskilling framework shows how external tools separate workers—

and students—from the creative intellectual labor that constitutes expertise. These pedagogues warned about alienating educational structures decades before AI existed in classrooms.

Yet these two scholarly conversations have not intersected. No research systematically applies Marx's alienation theory to analyze how AI automation creates pedagogical alienation in the cognitive labor through which programming expertise develops. We lack empirical investigation of how different forms of AI mediation affect students' relationships to their intellectual work. We cannot distinguish automation that liberates from automation that alienates without rigorous theoretical frameworks grounded in empirical study.

This gap is urgent. As AI tools proliferate in programming education, we risk producing a generation of technically competent but intellectually dependent professionals—programmers who can prompt but cannot design, who consume solutions but cannot create them. The next chapter develops a framework of pedagogical alienation that makes this transformation both analyzable and empirically investigable—providing the theoretical tools necessary to understand and address AI's impact on programming education.

# 4 Theoretical Framework: Pedagogical Alienation

Marx first described his conceptualization of alienation in the mid nineteenth century and applied it to the analysis of workers under industrialized capitalism. The alienation framework has proven beneficial in understanding the relationships of individuals to the power structures they inhabit. Scholars and theorists have extended the concept to analyze such relationships in domains beyond industrial revolution era capitalism, such as office workers and educators. This chapter explores the foundations of alienation and its extension to these other domains. Then the concept of alienation is extended to the GenAI enhanced classroom and to programming education. Finally, the indicators of GenAI mediated alienation are described so that we can determine the extent to which students become alienated from their own learning.

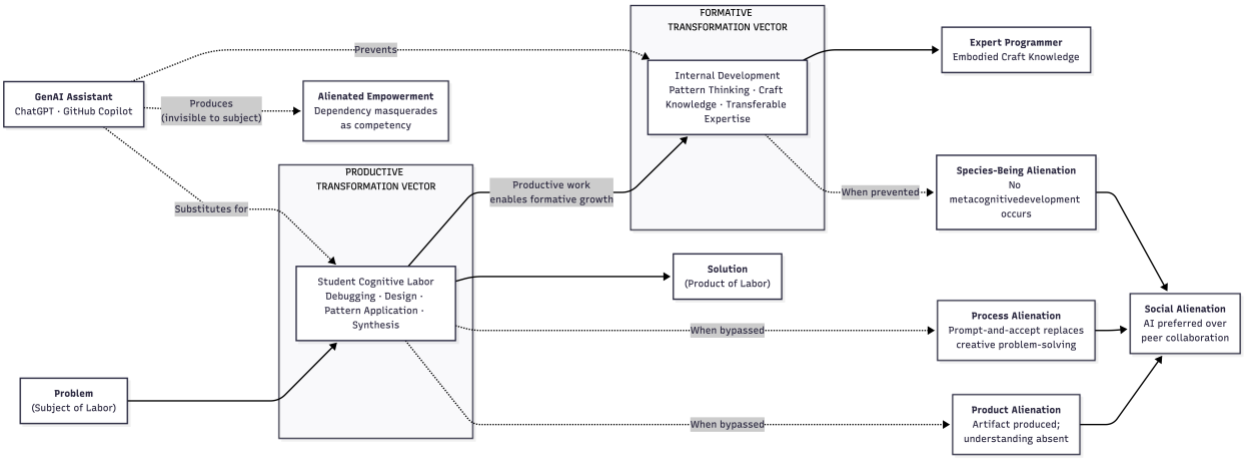


**Figure 4.1: The Dual Transformation Framework and Forms of Pedagogical Alienation in AI-Assisted Programming Education**

## 4.1 Programming Education as Cognitive Labor

When originally presented, the alienation framework was used to describe the experience of workers in factories who created physical goods. The terms were clear and it was easy to see—literally—the alienation occurring. In the twentieth century when workers traded in their hammers for computers, alienation took on new forms and the framework had to be extended.

Before pedagogical alienation can be described, we must trace the development of alienation from its industrial origins, through the white collar office, and into labor of learning.

### 4.1.1 Cognitive Labor and Its Alienation: A Historical Foundation

In 1832, Charles Babbage—pioneer of programmable computing—observed that "the division of labour can be applied with equal success to mental as to mechanical operations" [48]. This recognition that cognitive work constitutes labor subject to economic analysis predates Marx's systematic analysis of labor as the purposeful activity that transforms a *subject of labor* into a *product* possessing new *use-values*. As Marx writes in *Capital*, "human beings use means of labor to alter an object of labor, working from the start with a specific purpose in mind. The process vanishes in the product. Its product is a use-value, a piece of natural material whose form has been changed to make it suitable for satisfying human wants or needs" [49]. Babbage's insight is that mental operations—analysis, calculation, planning—constitute labor in precisely this sense: they transform raw information (subject) into useful knowledge (product) through purposeful cognitive effort (labor). The invisibility of cognitive labor does not negate its reality or intensity. On the contrary, abstraction *increases* cognitive load—as Detienne demonstrated, programmers must simultaneously think in abstract models and concrete implementations, a dual-processing demand absent from most physical labor [2].

Workers who engage primarily in cognitive labor are subject to the same threat of alienation as manual laborers. Even though cognitive processes are intangible, they are commodified and appropriated by capital: "the intellectual faculties involved in the production process become completely separated from manual labor…and now these faculties are fully transformed into powers that capital uses to control labor" [49]. The purpose of mental labor is the production of new forms of knowledge. This may be the discovery that a certain molecule

alleviates disease symptoms, the production of a commercial jingle, or the plan to deliver a product. When the worker produces this knowledge they do not own it and the conditions under which it is produced are not in the worker's control.

By the mid-twentieth century the nature of work had shifted dramatically. Many workers now created intangible products—plans, analyses, designs, strategies—through cognitive labor. Much of this type of work was akin to craft—productive work requiring tacit knowledge during which a laborer exercises creative judgment over the design of the product and the process through which it is crafted. In the modern workforce, however, these craft skills are broken down through the division of labor into smaller parts that can be performed by less skilled workers meaning that "that the individual does not carry through the whole process of work to its final product; but it also means that under many modern conditions the process itself is invisible to him" [50]. This is alienation occurring in the realm of mental labor. Braverman analyzed the process through which the division of labor was applied to clerical work specifically, noting that, while it was once a job requiring general knowledge and education, management had systematized and standardized the work to the point where it was transformed into manual labor [51]. Work is routinized such that workers become easily replaceable—their day-to-day duties being repetitive tasks that require minimal skill.

The advent of the office machine caused further alienation of a mental laborer from the product of their work. For example, in the 1940s and 1950s the work of data processing was approaching a craft, even requiring an extended apprenticeship, but the invention of the computer quickly made this craft obsolete [51]. The human work of data processing changed from the mentally stimulating work of calculating solutions and analyzing data, into the monotonous task of data entry. These machines caused workers to become "estranged from the

intellectual potentialities and aspects of work" while the work of "each individual is routinized in the name of increased and cheaper per unit productivity" [50]. The purpose of the office machine is not, like the machinery in the factory, to control the motion of the work (e.g. screwing screws), but rather to control the flow of information [51].

Braverman observed one area resistant to this deskilling trend. In 1974, he noted that computer programming remained craft work, with programmers and systems analysts occupying 'the upper level…requiring specialized technical skills in people who can grasp the rationale of the systems they work on' [51]. This craft character has persisted: programmers exercise creative judgment over their work, see their contributions in the final product, and develop new skills through the work itself. The complexity of software development requires engineers to understand both discrete features and the system context as a whole—precisely the integration of conception and execution that defines craft labor.

This tradition—from Babbage's recognition that mental operations constitute labor, through Marx's analysis of commodification, to Mills' and Braverman's documentation of white-collar work's transformation—establishes that cognitive labor is subject to the same economic forces as physical labor. Educational contexts, however, introduce distinctive characteristics that require theoretical extension.

### 4.1.2 Learning as Productive Labor

Having established cognitive work as labor, we now examine educational labor specifically. The product of this labor, the purpose of education, is to produce understanding in the mind of the learner. This understanding is not just the acquisition and speedy recall of facts, though this may be a part of understanding, but rather the ability to apply internalized knowledge in future acts of labor. Consider a computer science student who comes to understand the

underlying structure of computer programming languages. This student sees not just the syntax, but also the abstract logic of the program itself—that the program is an idea expressed in syntax, not defined by syntax. This understanding enables the student to learn other programming languages with rapidity and indeed to program more efficiently in any programming language. This student has truly learned, their knowledge has a distinctive use-value to them, and they can apply this knowledge in scenarios dissimilar to the environment in which they acquired it.

In the context of education, labor is the set of actions that transforms information into transferable understanding. Unlike the other forms of mental labor where the product is knowledge that can be shared or given to others, the purpose of educational labor is to produce understanding in the mind of the learner. A student cannot create understanding for someone else, nor can someone else's understanding belong to them. This makes educational labor distinctive—the student is, at once, the *subject*, the *laborer*, and the *product*. Mills recognized that craft work develops not just skill but the person: work becomes 'a means of developing himself as a man…the cumulative result obtained by devotion to and practice of his skills' [50]. Educational labor makes this self-development explicit and intentional. The student labors not to produce commodities for exchange but to transform themselves through practice.

The process by which this understanding is created in the mind of the learner is a peculiar act of dual transformation. A student transforms themself—literally, learning creates physical changes in the brain—through the repeated transformation of problems into solutions. This is true for all forms of education though the nature of the problems varies greatly across disciplines. In programming pedagogy these problems often take the form of writing software that adheres to predetermined set of requirements. Instructors give students problems for which the solutions lie outside the student's current understanding, but that reinforce the concepts taught in the course.

The student's task is to understand the problem as it is presented then devise and implement a solution. But the production of a working computer program is only part of the work that the student must perform. In fact, the purpose of the programming problem is to develop the understanding which the student can use to implement solutions to problems vastly different than those encountered in the classroom. The understanding cannot be separated from the student.

In workplace labor, the worker can be separated from their product—the boss provides raw materials, the worker produces widgets, the boss sells the product, the worker receives wages. The worker may see their creation boxed and shipped away, containing no expression of their creativity or judgment. Mental labor exhibits the same separation: the analyst produces reports consumed by management, the programmer writes code owned by the company, the designer creates campaigns sold to clients. In each case, the product exists independently of its producer.

In educational labor, however, this separation is impossible. The product is the student transformed who possesses understanding that cannot exist apart from the one who created it. When the student is separated from the labor that produces understanding this is not mere alienation. In fact, when AI generates solutions—solutions which the student should create through their own cognitive labor—it not only alienates students from the product they created; it alienates them from learning itself, preventing the self-transformation that *is* the product of educational labor. The student receives functioning code yet remains unchanged—no understanding produced, no development achieved, no self-transformation.

This dual transformation creates two interdependent vectors through which alienation can occur. The *productive transformation vector* involves the external work students perform—transforming problems into solutions, requirements into code, questions into answers. The

*formative transformation vector* involves the internal development that occurs through that work—transforming novice into expert, uncertainty into understanding, potential into capability. These vectors are causally linked: the formative transformation occurs *through* the productive transformation. Disrupting the productive vector necessarily prevents the formative vector.

Educational alienation is thus preventative rather than possessive. In workplace alienation, workers are separated from products they created—the widget exists, just belongs to the boss. In educational alienation, students are prevented from creating the product (understanding) in the first place. They cannot be alienated from understanding they possess; one either understands or does not. Rather, they are alienated from the productive transformation through which understanding develops. When AI performs the productive work, students receive artifacts (code) without undergoing formative transformation (learning).

Programming education exemplifies this distinctive character of educational labor in particularly clear form. Pattern thinking exemplifies this integration: students develop expertise only by repeatedly transforming problems into solutions, and themselves in the process.

### 4.1.3 Pattern Thinking as Craft Knowledge

Programming expertise is craft knowledge. Like traditional crafts, it cannot be transmitted through instruction alone but must be cultivated through sustained practice. Just as a master woodworker sees in a stack of raw lumber the sturdy oak table, so too does the expert programmer see a functioning application from a set of primitive functions and library code. The expert can simultaneously work forward from raw materials to the finished product, and backward from the desired outcome to the required pieces—they have the ability to hold in their mind at once the shape of the entire project from conception to result. One cannot simply read in

a textbook how to do this. A woodworker must know how the grain of the wood affects the cuts of the saw; a programmer must know how a data structure affects the processing of data.

Like the woodworker whose craft is not merely cutting wood but shaping it to purpose, the programmer's craft is not writing code but managing complexity. Whether working with greenfield or brownfield code, a programmer must create a functioning application that solves real-world problems. To do this effectively requires pattern thinking—the ability to recognize recurring problem structures and deploy proven solution templates. This expertise, like all craft knowledge, develops through practice rather than instruction. As Baltes et al. demonstrated, programming expertise emerges through sustained deliberate practice, monitoring, feedback, and self-reflection. Pattern thinking cannot be taught; it must be cultivated [17].

To learn to detect and apply patterns is to learn a different way of thinking about problems. This is tacit knowledge that requires experience—experience that reveals what does not work just as much as what does work. Expert programmers have vast mental libraries of existing "programming plans" or patterns that they use to construct complete solutions to complex problems [4]. These patterns are snippets of code (e.g., 'search for item with loop', 'mutate all items in a collection') or formal design patterns (e.g., 'observer', 'state') that can be sequenced in a multitude of ways and adapted to the specific task at hand. Because the situations where these individual pieces apply overlap (should one use a mapping function or mutate data in place), programmers develop heuristics to guide their practice, but cannot write rules applicable to all situations. A programmer must be as familiar with what does not work as they are with what does work, they must know when to code against the heuristic, and resist design elegance when it is premature optimization.

When the novice programmer begins their learning journey, they see only the syntax—the loop construct, the if-else block—not the underlying flow of the program's logic or the interactions among the components. The expert looking at the same code sees the patterns of iteration and mutation—they think in terms of *semantics* not *syntax*. Knowing the syntax of a programming language makes you no more an expert than a woodworker who knows only the names of different types of saws. To develop pattern thinking, it is not enough to memorize syntax and pattern definitions. Instead, pattern thinking comes from encountering pattern implementations in various contexts—GUI frameworks, event systems, data pipelines—until the abstract principles crystallize. A similar process occurs in the development of expert pattern application skills. Expert programmers have implemented patterns correctly and (maybe more importantly) incorrectly countless times. This transformation from syntax-thinking to pattern-thinking occurs not through instruction but through *doing*—through writing code, encountering failures, and discovering what works in context.

This parallel between pattern thinking and physical craft is not decorative metaphor, but precise. Where the woodworker develops a 'feel' for grain, tension, and balance through extended hands-on work in the shop, a programmer develops a deep intuition for coupling, cohesion, and abstraction through minds-on engagement with code. Both forms of expertise are *embodied* (cognitive or physical), *practice-dependent* (cannot be transmitted through instruction alone), *judgment-based* (require contextually aware decision making, not rule application), and *iterative* (develop through cycles of attempt, feedback, reflection, and refinement).

Because this tacit knowledge requires sustained practice and experience it is particularly vulnerable to AI-mediated alienation. In craft disciplines, the productive and formative vectors collapse: the act of creating the product IS the process through which expertise develops. A

woodworker learns by making tables; a programmer learns by writing code. When AI automates this productive work, it doesn't merely separate students from their products—it eliminates the formative transformation entirely, because in craft, doing IS learning.

When the apprentice programmer is able to prompt an AI agent for the solution to a programming problem, they deprive themselves of the cultivation of intuition. Pattern thinking develops only by *thinking* through a problem, deliberately applying a pattern as a solution, and evaluating the effectiveness of that pattern. If the pattern does not work well, the apprentice must reevaluate and try a different approach. This process of apply, test, evaluate is critical. A woodworker's apprentice learns very little by watching the master cut joints; only when they make the cut themself, fail, adjust, and adapt do they develop the craft. Just as a machine that cuts joints for the apprentice prevents them from learning how to hold the saw, how fast to move the blade, and how to adjust when encountering a knot, AI that generates pattern implementations prevents students from learning when to create a new abstraction, when to apply a different pattern, and when to refactor existing code.

### 4.1.4 Automation as Cognitive Displacement

Programming has always involved tools—compilers that detect syntax issues and logic bugs, IDEs that provide predictive code completion, debuggers that reveal the state of a running program. But these tools require an engaged programmer who understands the program's code and why it is built. They augment the practice of building software by enabling programmers to type less, catch errors sooner, and debug easier. A programmer using a debugger to investigate the current state of a running program, is able to glean more information about the application than with logging alone, giving them more information to process in aid of diagnosing and

solving a problem. These tools augment cognitive labor—making it easier, faster, more efficient—but they require programmers to perform that labor.

AI coding assistants like ChatGPT or GitHub Copilot are a qualitative shift in the practice of programming. Instead of augmenting the work that a programmer does, they substitute it. A programmer making heavy use of GenAI need not understand the surrounding code context, nor hold in their head a model of the program, nor consider the runtime state. Instead one only needs to tell the GenAI tool that a bug exists and in what part of the code and it will investigate, diagnose, and attempt to fix the issue. The process of walking through the code with the debugger, comparing the current state to the expected state, and determining the cause of a problem is replaced by the action of typing a prompt into a chat window. One might argue that GenAI still requires cognitive engagement—that students must understand their code well enough to formulate effective prompts. But this defense fails. GenAI coding agents process entire codebases and work from minimal natural language descriptions. A student can describe a bug in plain English (‘my code isn’t compiling’) without understanding why it’s failing or what the fix requires. The AI investigates the codebase, diagnoses the issue, and implements a solution—cognitive work the student never performs. The student need only recognize that something is wrong and roughly where—understanding *why* it’s wrong or *how* to fix it is unnecessary.

Prior to GenAI, predictive auto-complete was the most advanced form of code generation available. It works by suggesting possible completions—variable names, function calls—within a text editor’s interface. The programmer selects the correct one from the available completions. These tools are quite sophisticated and can suggest symbols from within the entire project and all the dependent libraries, only revealing to the programmer those symbols that are available within

the scope they are currently typing and the object they are currently referencing. But the programmer still has to know what purpose each symbol serves. For example, a developer types the name of a variable representing an object of a specific class type. The IDE displays a list of all possible functions that can be called on this object and the programmer selects the function and function signature. To make the correct selection the programmer must know the context of the current object, possible states that it may have, available data to be passed as arguments to the function. They must handle the return value of the function and account for any internal state changes to the object after the function call. Predictive auto-complete is doing much of the typing, but the programmer is doing all of the thinking.

GenAI also generates code, but it does so with nothing more than a prompt from a programmer. Consider a common scenario: a student struggling with an assignment to write a binary search tree using an array as the underlying primitive data structure and an in-order tree traversal algorithm. They have spent some time adding print statements to trace the path the algorithm is taking through the tree, and they discover that their implementation is not traversing the tree, but rather iterating through the array cell-by-cell. Instead of turning on the debugger and tracing the execution of the code they wrote—coming to understand how their current implementation is behaving and then comparing that to the expected behavior—they turn on Copilot. They prompt Copilot with “Implement an in-order traversal function on this class definition.” Copilot reads the open text file, implements the function, creates a test harness, runs the code to determine if it is correct, fixes any issues that it encounters, and then reports to the student what it has done. In this scenario the student has automated their thinking. They do not experience the process of mentally tracing the execution of a program, they do not learn how to model programs in their mind, they do not learn what steps to take to fix things when they break.

They do not gain the tacit knowledge of how programs work, of the effects of their decisions on a program's execution, of how to modify existing code to change its functionality.

The purpose of these assignments is to create the desirable difficulties and productive failure that create transformative learning. Students *should* find assignments difficult—if the assignments were within the student's understanding learning would not occur. The cognitive labor of a student is to transform facts and information into transferable knowledge and skills. For programming students, this transformation occurs through the work of transforming problems into solutions. Educational labor has a distinctive property: the worker and the product are the same entity. When AI transforms problem → solution, it prevents the transformation of student → expert. The student receives working code but loses the cognitive development that is the point of education.

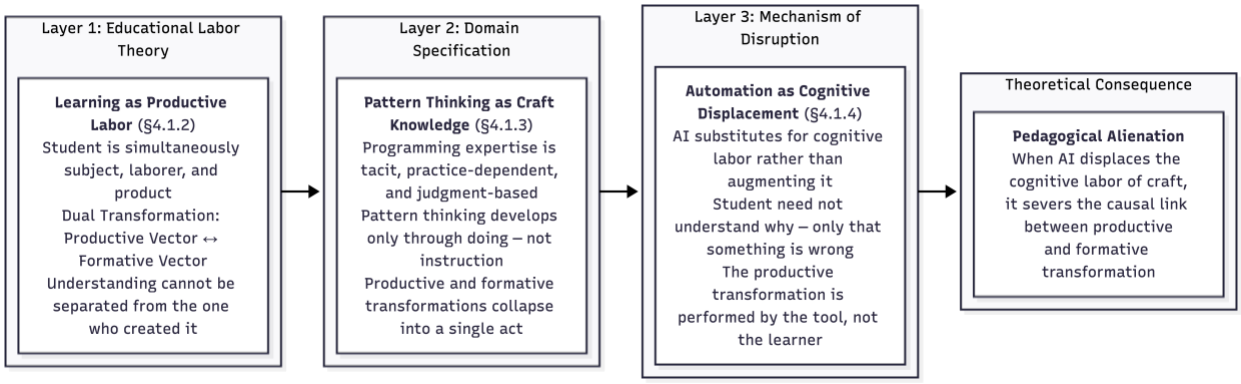


**Figure 4.2: A Layered Model of Cognitive Labor, Craft Knowledge, and AI Disruption in Programming Education**

## 4.2 The Four Types of Pedagogical Alienation in AI-Assisted Programming Education

Marx identified four types of alienation in industrial labor: product alienation, process alienation, species-being alienation, and social alienation. In educational contexts, these manifest distinctively because education involves dual transformation (Section 4.1.2): the productive

transformation of problems into solutions AND the formative transformation of students from novices into experts. This dual transformation means alienation can occur along two vectors—students can be separated from their productive work (creating solutions) or from their formative work (developing expertise), or both. This section examines each alienation type's manifestation in educational labor generally, then analyzes how programming education—as cognitive craft requiring sustained practice—creates specific vulnerabilities to AI-mediated alienation.

Programming expertise requires two related but distinct capacities: pattern recognition and pattern thinking. Pattern recognition enables programmers to identify which patterns exist in unfamiliar code—seeing that a Factory is being used, spotting an Observer in a codebase. Pattern thinking is the creative capacity to select appropriate patterns for novel problems, understand their trade-offs, transfer them across domains, and synthesize them into elegant solutions. Recognition is necessary but insufficient; thinking is the craft itself. Each type operates along productive and formative vectors, but manifests differently depending on the domain's learning requirements and the tools that mediate them. AI tools may aid pattern recognition while impeding pattern thinking—showing students what patterns look like while preventing the cognitive labor through which pattern thinking develops.

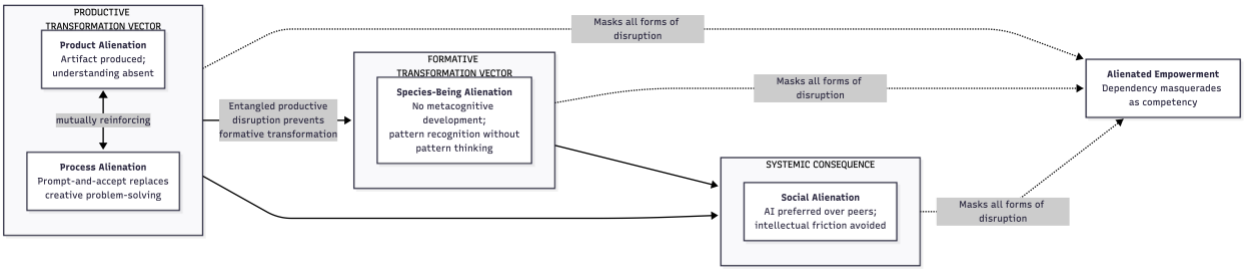


**Figure 4.3: Pedagogical Alienation: Relationships and Consequences**

### 4.2.1 Product Alienation

In the industrial capitalist mode of production, the worker transforms raw materials into a product with a new use-value. The employer takes the product from the worker and dispenses with it according to their purposes and thus "the worker is related to the *product of his labor* as to an *alien* object" [1]. For knowledge workers, though the product is intangible, the effect is the same. These laborers transfer ownership of their knowledge to their employer, retaining no control over its use and often receiving no acknowledgment for its production. In programming, not only is the code a programmer writes a product, but also the processes and architecture that are created in service of developing an application.

As we have established, the product of education is a transformed student with new understanding. But it is illogical to say that one is alienated from their own understanding—one either understands or one does not. In education, product alienation occurs on the productive transformation vector. The literature student purchasing an essay, the mathematics student copying a proof, the computer science student writing a prompt—they are all interrupting the productive transformation and denying themselves the formative transformation, receiving artifacts without developing understanding.

Programming's craft nature makes this product alienation particularly harmful as a student works to develop craft thinking. To learn to program requires not only mastering syntax and language constructs, but also developing the ability to think in abstractions, decompose problems, assemble elegant solutions, apply patterns, and use code to communicate intent and structure. When a student fails to understand their product, while they still complete the assignment, the formative transformation does not materialize and they do not develop the skills required for becoming a master craftsperson.

### 4.2.2 Process Alienation

The second type of alienation that Marx describes is alienation from process. This occurs when a worker does not control or meaningfully engage with the manner in which the product is made. The main mechanism by which such alienation manifests is the division of labor—the breaking apart of a complicated task requiring a skilled worker into discrete subtasks which require little to no skill and can be performed by most people. For example, the job of the master woodworker who transforms raw lumber into a beautifully wrought table with ornate carving and a fine stain is replaced by multiple workers in an assembly line who repeatedly perform a single task such as cutting lumber, gluing joints, and applying stain to create hundreds of copies of the same table. The worker is thereby alienated from the *process* of creating the table as a whole and may not understand or have knowledge of the steps performed before or after their own work.

The craftsperson is a worker who is not alienated from their work process. On the contrary, Mills describes "craftsmanship as a fully idealized model of work gratification," and he argues that there are six components required, among which are the lack of an ulterior motive other than the creation of the product, the fact that the daily tasks are linked in the worker's mind to the product of the work, the worker decides and controls the process through which the product is created, and that the craftsperson is able to learn from their work [50]. None of these factors are present when the division of labor has reduced a worker's daily job to simple repetitive tasks that require little in the way of mental stimulation.

In the educational setting, process alienation can manifest on both vectors of the dual transformation process. In the productive transformation students may be alienated from the process when using AI by distancing themselves from the actual work itself. Just as an industrial laborer engages with the work process by performing only a single task, the student using AI to create a solution may only engage by prompting the AI itself. The final product is created, but

the student is not its creator—they direct the AI tool to create but do not participate in constructing and refining. Even if the student understands the solution and the reasoning behind it, their participation in the process is not complete as they perform only a small part of the work themselves.

The programming student who uses AI to complete an assignment may understand that the solution calls for the use of a design pattern, and they may direct the AI tool to use the pattern in its solution. They do not, however, write the code that implements the pattern themselves. The process of constructing their assignments transforms, as Shihab has noted, from a creative event to one of prompt and implement [31]. The student attempting to become a master bypasses the processes through which mastery develops.

During the formative transformation process, we can see learning occur through methods which the student has no control over or with which they do not meaningfully engage. The pressure to perform well and to compete with one's peers is ever present in higher education. Students who make use of AI on assignments and exams may achieve higher grades than those who do not. This incentivizes all students to use tools which they may not enjoy using or which they think interfere with their style of learning. Even though a student develops understanding through the use of AI, the fact that they feel compelled to use it instead of choosing to use it means that they are separated from their own learning process. They may experience their learning process as lackluster and mechanical.

When a programming student is using AI to complete an assignment, even though they understand the code that the AI has generated, they lack the satisfaction they would feel had they written each line of code themselves. Perhaps this student is able to explain what each line of code does, the rationale behind the design the AI created, and even suggest improvements. But

the very fact they did not create the entire program themselves from start to finish leaves them with a sense of disengagement.

### 4.2.3 Species-Being Alienation

A species-being is an organism that can conceive of itself as an individual member of a species, freely direct its activities without adherence to innate instinctual behavior patterns, make its own life the object of its activity, and develop its own capacities and contribute to the development of the species as a whole. Humans are species beings. Even when a human is performing an activity solely for its own survival (gathering food, cooking, rearing the young) it has the freedom to choose how to do that thing and creatively improve the process. Capitalist labor practices turn "*man's species being*, both nature and his spiritual species property, into a being *alien* to him, into a *means* to his *individual existence*" [1]. The unalienated human lives to work because their work is the manifestation of their being—the realization of their potential as a human—and this labor is intrinsically meaningful. The human alienated from their species being works only as a means to survival—their labor does not develop their capacities.

In pedagogy, species being alienation occurs when a student is performing the activities required of them but is not developing their species being capacities. The nature of our modern education system, with its focus on the future market value of skills and knowledge, encourages students to optimize for high grades and diploma acquisition over deep understanding and curious exploration. GenAI enables students to circumvent much of the deep thinking and trial-and-error that is part of the learning process. Instead of writing and rewriting a sentence until it clearly conveys what they intend, a student asks an AI to revise it for them—or worse, they ask the AI to write the entire paper on their behalf. When doing this, the student does not engage in higher order thinking—a human species being trait—and therefore does not develop their

capacity for such thinking. This prevents what Marx describes as "the *real*, active orientation of man to himself as a species being…only possible by his really bringing out of himself all the *powers* that are his as the *species* man" [1]. The student using GenAI is orienting themself away from their species being by outsourcing the exact cognitive labor that would develop these powers.

When a programming student attempts to develop pattern thinking, they are engaging in a species being activity: reflecting on their own thinking for the purpose of improving it. This capacity emerges only through the difficult work of repeatedly applying patterns in concrete software design challenges. When a student relies on GenAI to write a factory, implement an adapter, or apply a chain of responsibility, they bypass the cognitive labor involved in mapping the pattern to the specific problem—they are not thinking about their own thinking at all. While they may develop pattern recognition skills by seeing AI-generated examples, offloading the problem-solving process prevents the development of pattern thinking itself—the creative intellectual work through which programming expertise emerges. In this way, the student is estranged from the activities that cultivate their species capabilities.

### 4.2.4 Social Alienation

The consequence of product, process, and species being alienation is that humans are alienated from one another. Indeed, a person who is alienated from their own species being must necessarily be alienated from the species being of another. Marx understands this social alienation as relationships among workers wherein "each man views the other in accordance with the standard and the position in which he finds himself as a worker" [1]. In other words, the same way that a worker relates to the product of their work as something alien to them, so too do they relate to their peers as alien.

School is inherently a social institution. Students engage in relationships with their instructors and with their peers, and much of the benefit of schooling arises from the interplay of perspectives and ideas within the classroom. Students learn not only from the lessons taught, but also through discussing course content with peers and by seeking advice and counsel about what they are learning. These exchanges are fundamentally reciprocal: in explaining concepts to another person, students must respond to questions, misunderstandings, and alternative interpretations, a process that deepens their own understanding.

When GenAI enters the classroom, however, students increasingly turn to chatbots to discuss concepts and request alternative explanations of instructional material. Unlike peer dialogue, these interactions are largely non-reciprocal. The sycophantic proclivities of many GenAI tools mean that the user's ideas and understanding are often reflected back to them rather than meaningfully contested or reshaped. The issue, then, is not merely the use of GenAI itself, but the reconfiguration of pedagogical relations it enables. When students, guided by confirmation bias and disconfirmation avoidance, preferentially engage with agreeable GenAI systems instead of peers with independent perspectives and interests, they become increasingly alienated from one another as intellectual agents participating in collective inquiry.

This reconfiguration of pedagogical relations has particular consequences in programming education, where learning depends heavily on the development of pattern thinking rather than the rote acquisition of syntax or tools. Pattern thinking emerges through exposure to multiple approaches, imperfect explanations, and competing mental models of a problem. When students work with one another, they encounter alternative decompositions, divergent assumptions, and idiosyncratic strategies that force them to recognize underlying structures that

transcend any single solution. These encounters are not always efficient or affirming, but they are formative.

In contrast, when students rely primarily on GenAI systems for assistance, the opportunity for this kind of pattern thinking is diminished. Sycophantic or overly accommodating responses tend to converge on solutions that echo the way that a student framed the problem to the AI, reinforcing surface-level understanding rather than disrupting it. Without sustained student-to-student interaction, learners are less frequently confronted with unfamiliar patterns and less often required to reconcile competing conceptualizations. As a result, the social conditions that foster pattern thinking in programming are weakened, further intensifying pedagogical alienation by separating students not only from one another, but from the shared cognitive labor through which programming expertise develops.

These four types of alienation do not operate in isolation—they form a mutually reinforcing system particularly acute in programming's craft context. Product alienation (receiving code without understanding) enables process alienation (reducing creative work to prompting), which prevents species-being development (no higher-order thinking emerges from prompting), driving students toward social alienation (preferring agreeable AI over challenging peers). Each form of separation intensifies the others. In craft disciplines where productive work and formative transformation collapse into single activities, this systemic alienation becomes especially acute. The next section develops observable indicators for each type of alienation, establishing what evidence would reveal their presence in programming education.

## 4.3 Indicators of Alienation

This unity of producer-product makes educational alienation qualitatively different from industrial alienation—a point that becomes crucial when AI mediates the learning process.

Braverman criticized industrial automation because it creates a worker whose 'brain has been separated from its body, having been appropriated by modern management' [51]. AI in education creates a parallel dynamic: the cognitive labor of learning—the 'brain' of intellectual development—becomes separated from students and appropriated by automated systems. When a student uses GenAI they invert the relationship between themself and their tool—instead of the student using the tool to enhance their thinking, they become a tool that organizes the thinking of the AI.

Detecting alienation requires distinguishing student performance from students' relationship to their own learning—no direct correlation exists between the two. High-performing students may be profoundly alienated from the cognitive processes that produce understanding, while struggling students may remain intellectually engaged with their own development. The distinction between *what students produce* and *how they relate to that production* is methodologically critical. Each type of alienation that has been mentioned above will have its own set of indicators—along both the productive and formative vectors—that we can look for in students to determine the degree to which they are alienated from their own learning.

Product alienation (Section 4.2.1) manifests as separation between artifacts produced and understanding developed. The prime indicator is inability to explain one's work: students produce functionally correct solutions but cannot articulate reasoning behind design decisions. Critically, this manifests independently of grades—a student earning high marks while unable to explain their work shows extreme product alienation, while a student with poor grades who understands their knowledge gaps shows low product alienation. Performance-confidence gaps provide additional evidence: students perform well on assisted assignments but poorly on

independent assessments, predicting high exam performance based on assignment success then struggling when supports are removed. Evidence appears in exam performance, explanation quality, and self-assessment accuracy documented in journal reflections.

Process alienation (Section 4.2.2) manifests as separation from the creative work of problem-solving. Students reduce their engagement to prompting and directing AI rather than constructing solutions themselves—workflow transforms from design-implement-refine cycles to prompt-and-accept patterns. On the formative vector, students report feeling compelled to use AI due to competitive pressure despite preferring to work independently, experiencing their learning process as mechanical or unsatisfying even when they understand outcomes. Evidence includes workflow descriptions in journals, time allocation between thinking and prompting, and student reports of autonomy versus compulsion in tool use.

Species-being alienation (Section 4.2.3) manifests as separation from higher-order cognitive development. Students complete tasks and produce correct solutions but fail to develop metacognitive awareness or reflective thinking about their own problem-solving processes. They optimize for efficiency and grades rather than engaging in the productive struggle that cultivates deeper capacities. Evidence includes absence of self-reflection in journals, inability to articulate their own thinking processes, and development of pattern recognition without corresponding pattern thinking—students can identify patterns but cannot explain why particular patterns fit particular problems or make independent design decisions. Indicators appear in journal depth, metacognitive language use, and ability to justify design choices beyond surface recognition.

Social alienation (Section 4.2.4) manifests as preferential engagement with AI over peers. Students turn to chatbots for explanation and assistance rather than seeking collaborative problem-solving with classmates. They report finding AI interactions more efficient or

comfortable than peer dialogue, avoiding the intellectual friction that emerges when encountering alternative approaches and competing mental models. Evidence includes reduced peer collaboration, preference for AI assistance over study groups, limited exposure to diverse problem-solving strategies, and journal reports indicating AI as primary resource. Indicators appear in collaboration patterns, help-seeking behavior, and students' descriptions of their intellectual community.

**Table 4.1: Indicators of Pedagogical Alienation: Type, Transformation Vector, and Cognitive Cost**

| Alienation Type | Indicator | Transformation Vector | Manifestation | Cognitive Cost |
|---|---|---|---|---|
| **Product** | Cannot Explain Own Work (PA-1) | Productive | Student produces correct solution but cannot articulate reasoning or reconstruct it without assistance | Understanding never formed; formative transformation did not accompany the productive output |
| | Confidence-Capability Gap (PA-2) | Productive | High performance on AI-assisted work; significant underperformance on independent assessment | Self-assessment decoupled from actual capability; student cannot identify their own knowledge gaps |
| | Ownership Denial (PA-3) | Productive | Student explicitly attributes intellectual work to AI; experiences the product as "not mine" | No integration of new knowledge into the student's developing expertise or identity as a programmer |
| | Opaque Understanding (PA-4) | Productive | Student can use or apply a solution without being able to explain the principles underlying it | Pattern recognition without pattern thinking; surface familiarity without transferable understanding |
| **Process** | Prompt-and-Accept Workflow (PRA-1) | Productive | Student iterates prompts until acceptable output is produced without critically evaluating intermediate steps | Problem-solving process bypassed; iterative reasoning and design judgment never exercised |
| | Compelled AI Use (PRA-2) | Both | Student uses AI despite preferring independent work due to competitive pressure or time constraints | Autonomy over learning process lost; engagement experienced as external compulsion rather than self-directed inquiry |

| Alienation Type | Indicator | Transformation Vector | Manifestation | Cognitive Cost |
|---|---|---|---|---|
| | Reduced Problem-Solving Agency (PRA-3) | Productive | Student moves directly from problem statement to AI output without intermediate reasoning or decomposition | First-principles thinking not exercised; problem decomposition — the most cognitively generative phase — outsourced |
| | Mechanical/Unsatisfying Experience (PRA-4) | Both | Student reports the work as rote or intellectually disengaging despite successful completion | Intrinsic motivation and craft satisfaction eroded; learning becomes transactional rather than transformative |
| | *Preference for Independent Work (PRA-5)* | Both | *Student explicitly values working without AI; reports greater intellectual engagement when doing so* | *Counter-indicator: engagement with productive transformation preserved; formative transformation more likely to occur* |
| **Species-Being** | Absent Metacognition (SBA-1) | Formative | Reflections describe tasks performed without examining thinking processes; descriptive rather than reflective | Metacognitive capacity neither exercised nor developed; self-directed learning and deliberate practice impossible |
| | Grade/Efficiency Optimization (SBA-2) | Formative | Student explicitly prioritizes completion speed and grades over understanding | Deep learning replaced by surface performance; instrumental rationality forecloses the curiosity that generates expertise |
| | Pattern Recognition Without Pattern Thinking (SBA-3) | Formative | Student correctly identifies patterns but cannot justify selection, reason about trade-offs, or consider alternatives | Expertise ceiling reached at recognition level; transfer and synthesis cannot develop |
| | Dependency Awareness (SBA-4) | Formative | Student acknowledges reliance on AI for thinking they recognize they should perform independently | Awareness without behavioral change indicates structural dependency; metacognition present but not generative |
| | *Emerging Autonomy (SBA-5)* | Formative | *Student demonstrates developing independent design* | *Counter-indicator: species-being capacities actively developing;* |

| Alienation Type | Indicator | Transformation Vector | Manifestation | Cognitive Cost |
|---|---|---|---|---|
| | | | *thinking; reasons from principles without external validation* | *formative transformation proceeding* |
| | *Transfer of Understanding (SBA-6)* | Formative | *Student applies pattern thinking in new contexts or novel problems beyond those encountered in class* | *Counter-indicator: understanding has become generative rather than situational; genuine expertise developing* |
| **Social** | AI Preferred Over Peers (SA-1) | Both | Student explicitly prefers AI assistance over peer collaboration, study groups, or instructor engagement | Exposure to alternative approaches and peer critique eliminated; pattern thinking development impeded |
| | Reduced Peer Collaboration (SA-2) | Both | Peer interaction absent from student narratives where collaborative engagement would be expected | Diverse problem framings and competing mental models not encountered; pattern thinking development narrowed |
| | Avoidance of Intellectual Friction (SA-3) | Formative | Student avoids situations involving peer critique; prefers AI's convergent responses over peer contestation | Disconfirmation avoidance prevents the conceptual revision through which pattern thinking deepens and generalizes |
| | *Peer Engagement (SA-4)* | Both | *Student reports active peer collaboration; values peer critique; learns from alternative problem-solving approaches* | *Counter-indicator: social conditions for pattern thinking preserved; collaborative cognition active* |

*Italicized rows denote engagement indicators — counter-evidence of alienation. Full operationalization of all indicators, including valence coding and example passages, is provided in Chapter 5.*

## 4.4 Conclusion

What is at risk is the loss of craft in programming. With GenAI, programming will no longer require tacit knowledge developed through sustained practice. The work will be less meaningful as the programmer no longer produces the software themself—they will see less of

themselves in the final product. The machine will dictate the programmer's actions and the next steps that they take to complete the task. Since the machine is doing the programming, the programmer has no opportunity to learn through their craft and become a better craftsperson. The work itself becomes invisible, the end product an opaque result of a formulaic conversation with a GenAI agent, the development process requiring less human interaction. The programmer becomes alienated from their work, from their peers, from themselves. Having established theoretical frameworks and observable indicators, the next chapter details how this research operationalizes these concepts to empirically investigate AI's impact on pattern thinking development.

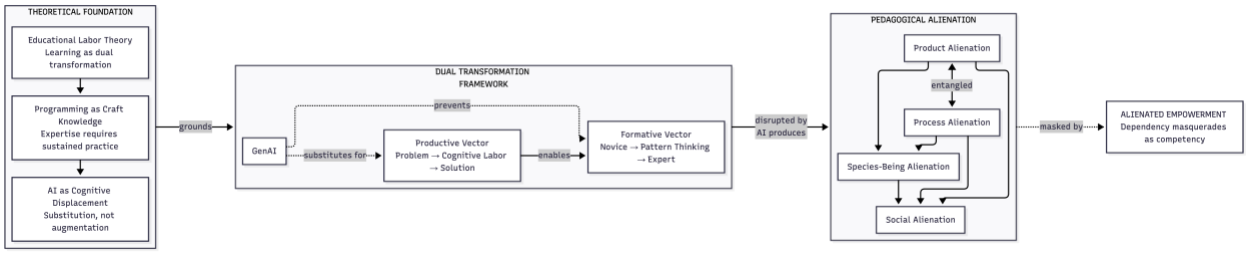


Figure 4.4: The Pedagogical Alienation Framework: A Structural Overview

# 5 Methodology: Measuring Alienation in Practice

Theory without evidence remains speculation. The alienation framework developed previously requires testing against classroom reality. Can product alienation be detected in exam performance? Does process alienation appear in journal reflections? This chapter describes the methodology used to make abstract theoretical constructs empirically investigable.

The research was embedded within ICS 372: Object-Oriented Design & Implementation, taught Fall 2025 at Metropolitan State University. Students learned design patterns under four experimental conditions with varying levels of AI assistance. Data collection included exam performance, weekly journal reflections, and AI interaction logs.

Section 5.1 explains applied philosophy methodology and its appropriateness for investigating pedagogical alienation. Section 5.2 describes the research context, participants, and four experimental conditions. Section 5.3 details how each alienation type was operationalized through multiple data sources. Section 5.4 addresses ethical considerations and IRB approval.

## 5.1 Applied Philosophy Methodology

This research employs applied philosophy methodology—using theoretical frameworks from critical theory to analyze empirical data from educational practice. Unlike typical computer science education research that measures learning outcomes to determine tool effectiveness, this study uses Marx's alienation theory to understand *how* AI tools transform students' relationships to their intellectual work. The research questions examine whether GenAI affects pattern recognition, application, and synthesis. But these questions serve a deeper epistemological and ethical inquiry: what does GenAI do to the learning process itself, and what are our obligations as educators when tools enable students to bypass the productive struggle through which expertise develops? Most existing research on AI in programming education focuses on

effectiveness metrics—does performance improve?—while leaving unexamined the fundamental questions of how learning occurs and whether technological mediation fundamentally alters that process.

Alienation manifests differently across types, requiring different evidentiary approaches. Process, species-being, and social alienation emerge primarily through qualitative data—journal reflections reveal workflow transformations, metacognitive depth, and social preferences that quantitative measures cannot capture. Product alienation, however, requires both: the gap between students' confidence (qualitative) and capability (quantitative) IS the evidence. When students report understanding but cannot demonstrate it independently, this disconnect reveals alienation from the product of their labor. Mixed methods allows detection of all four alienation types through data appropriate to each.

This research prioritizes ecological validity over experimental control by embedding the study within an actual course rather than conducting laboratory experiments. Alienation—particularly process alienation (feeling compelled to use AI) and social alienation (preferring AI over peers)—emerges from authentic educational pressures: real grades, actual deadlines, competitive dynamics among students. These conditions cannot be replicated in artificial laboratory settings where stakes are absent and behavior may not reflect genuine learning patterns. This approach sacrifices the ability to isolate variables and control confounds, introducing potential threats to internal validity that laboratory conditions would minimize. However, for research examining students' relationships to their intellectual work rather than merely measuring learning outcomes, authenticity matters more than control. The findings will reflect actual classroom dynamics where AI integration decisions must be made, rather than idealized conditions that tell us little about real educational practice. This trade-off—control for

authenticity—aligns with the research's goal of understanding AI's impact on learning as students actually experience it.

This methodology draws on critical pedagogy's tradition of examining how educational practices reproduce or challenge existing power relations. Scholars like Freire, Apple, and McLaren established frameworks for analyzing educational processes through critical theory, while Gereluk and Kenklies specifically applied alienation theory to educational contexts. However, these critical approaches rarely appear in computer science education research, which predominantly employs quantitative effectiveness studies. This research brings critical theory methods into CS education, extending the alienation framework to a domain—programming pedagogy—where it has not been systematically applied. The contribution is not inventing applied philosophy methodology, but deploying it in a disciplinary context that typically does not employ such approaches.

## 5.2 Research Design and Context

The course the research was conducted in is a junior level computer science course at a four year university. To take the course, students were required to have completed a sophomore level course on data structures and a discrete mathematics course. The course was hosted both in-person on campus and online with synchronous weekly lectures. The in-person dates were reserved for the proctoring of the pencil-and-paper midterms and final exam. The purpose of this course is teach object-oriented design principles such as SOLID, polymorphism, inheritance, and design patterns. By the end of the course we hope students have a grasp of how to create a software application from the requirements phase to deployment using object-oriented development best practices and the proper application of design patterns.

When the semester began, there were thirty-two students enrolled but five students dropped before the semester finished. The data from the students who dropped the course is not included in the analysis. ICS372 is only available to degree-seeking students who have been admitted into the Computer Science program, and students are expected to have experience programming in Java and familiarity with object-oriented concepts such as class and interface. The material and assignments were all pedagogically relevant and students who opted out of the study performed the same work as those who participated. Demographic information such as race, age, and sex are not considered. A casual show of hands survey on the first night of class indicated that majority of the students intended to be professional software developers upon graduation.

### 5.2.1 Experimental Conditions

The experiments were integrated seamlessly into the course material. The assignments and exams were used as assessments along with required journal reflection entries. The journal entries asked students to reflect on their learning, their approach to problem solving, and their experience completing assignments with and without artificial intelligence. They were designed to provoke metacognitive thinking and provided students with an opportunity to reflect on their learning as well as to provide insight into their experiences for the purposes of this study. The key conditions were four assignments on design patterns—observer, strategy, factory, and decorator—with various requirements regarding GenAI use. The assignment of specific patterns to specific conditions was operationally determined by course structure, but the four conditions collectively created varied contexts for examining AI's impact on learning. Exams, since they were in-person and on paper, were used to gauge student understanding of the course material

independent of tool use since the students were only able to reference a single sheet of notes during the exam and they did not have access to GenAI or the Internet.

The observer pattern assignment required students to use GenAI to complete it. The task was to create a simulated real-time stock trading dashboard. Beyond simply having a subject emit data to observers, students had to implement dynamic subscriptions, concurrent updates from different sources, selective notification filtering, error handling, and multi-threading. It was intentionally designed to be beyond the scope of the basic observer pattern to ensure the students could not copy code from the teaching material and had to build something they had not seen in class. Working code was expected. Students kept a log of their interaction with the GenAI tool including the prompts they used, the responses from the tool, and a critical assessment of the code the tool generated. Additionally, they wrote a report on their assessment of the effectiveness of the GenAI tool and how using it affected their approach to problem-solving.

By requiring AI use on the observer pattern assignment, we can see what students do when they struggle with concepts they're not familiar with. At this point in their academic career, we can surmise that most students are not experienced with concepts such as thread-pooling and real-time systems. Since successfully completing the assignment required incorporating these elements, the use of GenAI played an instrumental, rather than an ancillary, role in the programming process. This gives us an opportunity to see how GenAI affects desirable difficulty and productive failure—the assignment was designed to be beyond the scope of what was covered in class. Analyzing the data from this condition has the potential to show product and process alienation.

The strategy pattern assignment prohibited the use of GenAI. For this assignment students were tasked with implementing a dynamic pricing system for a ride sharing application.

The system had a base price function of distance, time, and base rate. It also needed to accommodate surge pricing which added a multiplier to the base rate as well as a weather based pricing scheme that added a percentage surcharge. Different geographic zones had different base rates and multiple pricing schemes could be active at one time. In addition to working code, students also submitted a learning reflection. The reflection asked them to describe their approach to structuring their implementation, detail the external resources they used that were not GenAI, explain the inflection points where they got stuck or made decisions, and discuss the alternative solutions they considered before choosing the one they did.

Since the strategy pattern assignment disallowed GenAI, students had to rely on the course material and external non-AI sources such as Internet resources, peers, or their instructor. Because the AI required condition came first, students had a point of reference against which they could contrast their non-AI assisted problem-solving process. The reflection portion of this assignment elicited metacognitive thinking (comparing non-AI assisted problem-solving to AI assisted problem-solving) which would not have been possible had the AI prohibited condition come first.

The factory pattern assignment permitted but did not require the use of AI. Students, however, were required to justify their decision about their use of AI. This assignment tasked students with creating a character creation system for a video game engine that supported multiple platforms. Each platform had specific technological constraints (e.g. memory or polygons) and performance requirements (e.g. frame-rate). The factories had to account for these complexities and allow for creating multiple character types dynamically during runtime. In addition to working code, students had to create a software design in a UML class diagram, a character creation flow diagram, and an error handling strategy.

For the students who chose to use AI they explained why they chose to use it, what benefits they thought they would gain by using it, how they would evaluate the code the AI generated, and what they would do to ensure they understood the AI generated code. For the students who chose not to use AI they were asked what gave them the confidence to tackle the complexity without AI, what alternate resources they used, how they filled their knowledge gaps, and what benefits they thought they would accrue by not using AI. By asking these questions we can see how students perceive the usefulness of AI and how they think it helps their learning. We also hope to understand what mitigations they employ to ensure they are learning the material instead of letting the AI do the work for them, or if they choose to not use AI why they think that using it impeded their learning. Since on this assignment they had the option to choose after having experienced completing an assignment with and without AI, we can ascertain what they think the affects of AI are on their own learning.

The assignment on the decorator pattern required students to use AI, but it tasked them with critically analyzing the AI solution and improving upon it. For this assignment, they were asked to create a cloud security policy enforcement system that included four basic resource types: virtual machines, database instances, storage buckets, and API endpoints. Each resource type could be secured by one or all of six security policies that could have conflicting requirements and performance implications. The assignment was performed in three phases. First, the students were to prompt a GenAI tool to create the entire system. They were to submit the prompt (and any subsequent prompts if they refined the code) and the generated code. Next they were tasked with analyzing the generated code and creating a detailed report on how well the code complied with the requirements and how well it implemented security best practices. The analysis was also supposed to include consideration of performance, error handling, and

policy conflict resolution. Finally after performing their analysis they were to improve the implementation based on the findings in their analysis. They were allowed to use AI in this final improvement phase.

This assignment also required a reflection of their experience. They reported on what the AI did well and what it failed to do. They were asked if they learned anything based on the analysis of the generated code and if they might have missed something important if they just used the code as-is. Then they discussed how and when to trust the code that AI creates. The purpose of this assignment is to see if the students understand the code that an AI writes, if they can critique it, and if they can see what needs to be improved. These skills are essential to developing software development expertise. This assignment can reveal the degree to which students understand the concepts as it applies to real working code and if they can detect misapplications of the patterns or missing requirements.

In addition to the four core assignments, there were two midterms and one final exam. The exams were useful in that they gave a controlled environment for testing mastery of the course material. Since AI was not available, students had to use their understanding of the material only. The exams for this course were practice based. They required creating design artifacts, writing Java code, and analyzing Java code to detect patterns. The tests did not contain any multiple choice or true/false style questions.

**Table 5.1: Experimental Condition Design: Assumptions, Considerations, Expectations, Structure, and Exceptions**

| Assignment | AI Condition | Assumptions | Considerations | Expectations | Structure | Exceptions |
|---|---|---|---|---|---|---|
| **A1: Observer Pattern** | Required | Task complexity (multi-threading, concurrent updates, dynamic subscriptions) would | Assignment was intentionally designed beyond lecture coverage to prevent copying from course material; AI interaction log required to | Primary site for product alienation (PA-1, PA-3) and process alienation (PRA-1, PRA-3); baseline for observing how | Working code + complete AI interaction log (all prompts, responses, and critical | S08's entire submission — including the interaction log — was AI-generated; S27 did not |

| Assignment | AI Condition | Assumptions | Considerations | Expectations | Structure | Exceptions |
|---|---|---|---|---|---|---|
| | | genuinely exceed students' current capabilities, making AI use instrumentally necessary rather than supplementary; students' individual patterns of AI engagement would become visible under mandated use | capture actual use behavior rather than relying on self-report alone | AI mediates the desirable difficulty the assignment was designed to create | evaluations) + written reflection on AI's effect on problem-solving approach | run their code before submitting; S04 found AI-generated code incompatible with their existing implementation; complexity occasionally exceeded students' ability to effectively direct the AI |
| **A2: Strategy Pattern** | Prohibited | Completing A1 first would give students a reference point for contrasting AI-assisted and independent work; removing AI would create a pure independent baseline; the comparison would be most metacognitively productive if AI-prohibited followed AI-required | Sequencing was deliberate — contrast with A1 maximized the reflective value of A2; reflection prompt designed to elicit metacognitive thinking (inflection points, alternatives considered) that A1's AI-mediated workflow could not produce; non-AI external resources remained permitted | Counter-evidence of alienation (PRA-5); independent pattern application baseline; evidence for the formative transformation argument — whether productive struggle without AI produces deeper understanding than AI-mediated completion | Working code + learning reflection addressing: approach to structuring the implementation, external resources consulted, inflection points and decision moments, alternative solutions considered before choosing the final approach | No process alienation indicators surfaced in descriptions of A2 — students uniformly reported greater intellectual engagement, not less; A2 became the primary reference point against which students articulated the costs of AI mediation in A1 |
| **A3: Factory Pattern** | Optional + Justification Required | Having experienced both required and prohibited conditions, students would make informed and reflective | Additional design artifacts (UML class diagram, flow diagram, error handling strategy) required beyond working code, testing higher- | Species-being alienation evidence through students' stated rationale for AI use or avoidance; self-selection patterns | Working code + UML class diagram + character creation flow diagram + error handling | Multiple students submitted no justification file (scored 0/4) — analytically distinct from a weak justification |

| Assignment | AI Condition | Assumptions | Considerations | Expectations | Structure | Exceptions |
|---|---|---|---|---|---|---|
| | | choices about AI use; the required justification would reveal how students conceptualize AI's effect on their own learning; students' choices would be stable indicators of their alienation profiles | order design thinking; justification was scored (0–4 points) with different prompts for AI-users vs. non-users; the decision itself was treated as a data point distinct from the outcome | revealing individual relationships to AI mediation; student perceptions of how AI affects their own learning and capability development | strategy + AI decision justification document (rationale for choice; for AI users: how they evaluated and validated output; for non-users: alternative resources used and benefits anticipated) | (1–2/4); students who chose not to use AI often did not formally document that choice; some students' stated rationale for AI use ("gaining familiarity with the pattern") revealed dependency they did not recognize as such |
| **A4: Decorator Pattern** | Required + Critical Analysis | Students who had developed genuine pattern thinking across A1–A3 would be capable of meaningfully critiquing AI-generated solutions; critical analysis of AI output could partially substitute for direct implementation in terms of formative transformation; the three-phase structure would force engagement beyond prompt-and-accept behavior | Domain complexity (six security policy types, four resource types, conflicting requirements) was designed to ensure the AI solution would contain identifiable flaws requiring genuine understanding to detect; the third phase (improvement) tested whether students could direct AI with substantive independent judgment rather than iterative prompting; the structure was an explicit attempt to preserve formative transformation when AI | RQ3 evidence (pattern synthesis capability); differentiation between students with genuine independent pattern thinking and those without — only the former would produce substantive critiques; test of whether structured critical engagement with AI output develops the synthesis understanding it was designed to assess | Phase 1: Initial AI prompt(s) and generated code; Phase 2: Detailed critical analysis report (requirements compliance, security practices, performance, error handling, policy conflict resolution); Phase 3: Improved implementation with documented modifications; written reflection on AI capabilities | The assignment revealed that meaningful critique of AI-generated code requires the same independent understanding the assignment was designed to develop — students without pattern thinking could not produce substantive analysis regardless of the structured scaffolding; S10 did not submit; superficial |

| Assignment | AI Condition | Assumptions | Considerations | Expectations | Structure | Exceptions |
|---|---|---|---|---|---|---|
| | | | performs the productive transformation | | , limitations, and effect on learning | critiques described rather than evaluated the AI output |

### 5.2.2 Timeline and Sequencing

The study spanned a 16-week semester (August-December 2025). Design patterns were taught in pairs, followed by individual assignments under different AI conditions and subsequent assessment. Students completed Observer and Strategy assignments (weeks 3-4) before Midterm Exam 1 (week 5), then Factory and Decorator assignments (weeks 7-8) before Midterm Exam 2 (week 10). The final exam (week 16) provided comprehensive assessment of all four patterns. This compressed timeline ensured students experienced all four AI conditions within the first half of the semester, with the remainder of the course building on pattern thinking foundations.

**Table 5.2: Assignment Schedule**

| Week | Pattern | AI Condition | Data Collection |
|---|---|---|---|
| 2 | Observer & Strategy Pattern Lecture | - | - |
| 3 | Observer Assignment | AI Required + Documentation | Assignment, Journals, AI logs |
| 4 | Strategy Assignment | AI Prohibited | Assignment, Journals |
| 5 | Midterm Exam 1 | - | Pattern recognition & application |
| 6 | Factory & Decorator Pattern Lecture | - | - |
| 7 | Factory Assignment | AI Optional + Justification | Assignment, Journals, (AI logs if used) |
| 8 | Decorator Assignment | AI Required + Critical Analysis | Assignment, Journals, AI logs |
| 10 | Midterm Exam 2 | - | Pattern recognition & application |
| 16 | Final Exam | - | Comprehensive assessment |
| Ongoing | - | - | Weekly journal reflections (14 total) |

This design ensured students experienced all four AI conditions before comprehensive assessment, with journal reflections capturing their evolving relationship to AI-assisted learning throughout.

## 5.3 Measuring Pedagogical Alienation

Theoretical frameworks require translation into practice. Section 4.3 identified what alienation looks like—the observable patterns that distinguish engaged learning from automated production. But identifying indicators is not the same as measuring them. How does one detect the gap between confidence and capability? What constitutes metacognitive depth versus surface reflection? This section describes the transformation of abstract concepts into concrete data collection and analysis. Three instruments captured different dimensions of students' experiences: exams revealed what they could do independently, journals documented how they experienced their work, and AI interaction logs showed what they actually did. Together, these sources made pedagogical alienation empirically investigable.

### 5.3.1 Data Collection Instruments

Three instruments worked together to capture pedagogical alienation across its different manifestations. Examinations revealed what students could do independently. Journals documented how they experienced their work. AI logs showed what they actually did when using tools. Each instrument targeted specific alienation indicators while providing triangulation points for others.

**Examinations.** Three paper-based, in-person exams assessed students' independent capability without AI assistance. This design was deliberate: product alienation manifests precisely in the gap between assisted performance and independent capability. Midterm Exam 1 (Week 5) came after students completed both the Observer pattern exercise with required AI and

the Strategy pattern exercise with prohibited AI. The exam tested whether students could recognize these patterns in unfamiliar code and apply them to novel problems without external supports. Students who performed well on the AI-assisted Observer exercise but struggled with Observer questions on the exam revealed product alienation—they had submitted working code without developing genuine understanding. Midterm Exam 2 (Week 10) followed the Factory and Decorator exercises, again testing independent capability after varied AI conditions. The Final Exam (Week 16) required synthesis across all patterns, revealing whether students had developed transferable pattern thinking or merely accumulated exposure to pattern implementations. All exams included both implementation tasks (write working code) and explanation tasks (justify design decisions). Students who could implement but not explain demonstrated the clearest evidence of product alienation.

**Reflective Journals.** Fourteen weekly journal prompts captured students' evolving relationships to their intellectual work. The timing of journal entries mattered: prompts came immediately after key learning experiences, capturing fresh reflection before memory faded or rationalization set in. Journal Entry 2 (due before Week 3 class) established baseline problem-solving approaches before formal pattern instruction. Journal Entry 3 (due before Week 4 class) came directly after completing the Observer exercise with required AI assistance, asking students to examine their relationship to the code they produced: "Could you recreate this solution from scratch without AI assistance? What parts feel like your intellectual work versus the AI's work?" These questions targeted product alienation while it was still viscerally present. Journal Entry 4 (due before Week 5 class) asked students to compare their Observer experience (AI required) against their Strategy experience (AI prohibited): "In which exercise did you feel more intellectually engaged with the actual process of finding solutions?" This direct comparison

revealed process alienation—whether students experienced problem-solving as intellectual work or as prompt engineering. Later journals probed species-being alienation through metacognitive questions about pattern thinking development and social alienation through questions about collaboration versus AI reliance. The journals provided longitudinal evidence of how students' relationships to their learning changed as AI mediation varied.

**AI Interaction Logs.** For exercises requiring or permitting AI, students documented their complete interaction histories. These logs made visible the work processes that journal reflections described. Exercise 3 (Observer, AI required) mandated comprehensive documentation: every prompt, every AI response, every evaluation, every implementation decision. This revealed whether students engaged critically with AI suggestions (evaluative "drifting") or simply iterated prompts until code worked (uncritical "shepherding"). Exercise 6 (Factory, AI optional) required justification documents explaining whether students chose to use AI and why. These decisions—and the reasoning behind them—revealed how students understood their own capabilities and relationship to automated assistance. Exercise 7 (Decorator, AI required with critical analysis) generated three-phase documentation: the initial AI solution, detailed critique identifying security and design flaws, and improved implementation with documented fixes. These logs provided behavioral evidence that could validate or contradict journal self-reports, capturing what students actually did versus what they believed they did.

**Table 5.3: Data Collection Instruments and Analysis Methods: Targeting Pedagogical Alienation Across Sources**

| Instrument / Method | Administration | Alienation Types Targeted | Primary Indicators Detected | Analysis Approach |
|---|---|---|---|---|
| **Examinations** | Three paper-based, in-person exams without AI access: Midterm 1 (Week 5, post-A1 and A2), Midterm 2 (Week | Product Alienation (primary); Species-Being Alienation (supporting) | PA-2: gap between assisted assignment performance and independent exam performance; implementation | Quantitative scoring of correctness; separate qualitative assessment of explanation and design justification |

| Instrument / Method | Administration | Alienation Types Targeted | Primary Indicators Detected | Analysis Approach |
|---|---|---|---|---|
| | 10, post-A3 and A4), Final Exam (Week 16, comprehensive synthesis) | | score vs. explanation quality score as distinct measures of understanding | quality; cross-exam performance trajectories revealing whether AI conditions produce durable or temporary capability |
| **Reflective Journals** | 14 weekly prompts timed immediately after key learning experiences throughout the semester; prompts designed to target specific alienation indicators at each stage | All four alienation types; longitudinal trajectory of students' evolving relationship to their intellectual work | PA-1, PA-3, PA-4 (ownership and explanation claims); PRA-1, PRA-2, PRA-3, PRA-4, PRA-5 (workflow and agency descriptions); SBA-1, SBA-2, SBA-3, SBA-4 (metacognitive depth); SA-1, SA-2, SA-3 (collaboration and help-seeking preferences) | Qualitative coding using full alienation indicator coding scheme; each coded passage assigned intensity (0–3) and valence (A = alienation indicator, E = engagement indicator); longitudinal pattern analysis tracking change across conditions |
| **AI Interaction Logs** | Submitted with A1 (AI Required), A3 (AI Optional), and A4 (AI Required + Critical Analysis); documented all prompts, responses, evaluations, and implementation decisions | Process Alienation (primary); Species-Being Alienation (supporting) | PRA-1: shepherding (iterative prompting without evaluation) vs. drifting (critical evaluation and deliberate modification); PRA-3: presence or absence of independent problem decomposition before prompting; SBA-3: critique depth as proxy for pattern thinking | Workflow pattern analysis (shepherding vs. drifting); time allocation between prompting and evaluation; quality of critique in A4 Phase 2 as independent measure of genuine comprehension; behavioral evidence used to validate or contradict journal self-reports |
| **Cross-Instrument Integration** | Post-semester systematic analysis after grade submission; primary vehicle for detecting Alienated Empowerment | All four alienation types; Alienated Empowerment (meta-phenomenon) | PA-2: divergence between journal confidence claims and exam performance scores; PRA-1: discrepancy between log behavior (shepherding) and journal self-report (claimed critical engagement) | Coherence and mismatch identification across instruments; triangulation of subjective experience (journals), demonstrated capability (exams), and actual behavior (logs); mismatches between self-report and performance treated as primary |

| Instrument / Method | Administration | Alienation Types Targeted | Primary Indicators Detected | Analysis Approach |
|---|---|---|---|---|
| | | | | evidence of Alienated Empowerment |

### 5.3.2 Analysis Methods

Analysis occurred in two phases: preliminary coding during the semester to track emerging patterns, and systematic analysis after grades were submitted to ensure research activities did not influence grading decisions.

**Qualitative Coding of Student-Authored text.** Journal entries, assignment reflections, AI interaction logs, and critical analyses were coded for indicators of each alienation type using a framework derived from Section 4.3. Product alienation manifested in students' inability to explain their own code or recognition of gaps between their confidence and actual understanding. Entries were flagged when students acknowledged they "couldn't recreate this without AI" or when they predicted strong exam performance but expressed uncertainty about explaining their solutions. Process alienation appeared in workflow descriptions—students who described moving from "prompt to implement" without intermediate problem-solving steps, or who reported feeling compelled to use AI despite preferring independent work. Species-being alienation emerged through metacognitive depth: entries lacking self-reflection about thinking processes, optimization for grades over understanding, or pattern recognition without corresponding pattern thinking (identifying patterns without justifying why they apply). Social alienation appeared when students reported preferring AI over peer collaboration or finding AI more efficient than working with classmates. Each journal was coded by alienation type and intensity (present/absent, mild/moderate/severe) with specific passages extracted as evidence. The full coding scheme is described below.

#### 5.3.2.1 Alienation Indicators Coding Scheme

Each coded passage receives:

1. **Code** — the alienation indicator (see below)
2. **Intensity** — 0 = absent, 1 = mild, 2 = moderate, 3 = severe
3. **Valence** — A = alienation indicator, E = engagement indicator (counter-evidence of alienation)
4. **Evidence** — the quoted passage

Engagement indicators (valence E) are equally important — they document where students resist or recover from alienation and are essential for tracing trajectories.

##### 5.3.2.1.1 Product Alienation (PA)

*Separation between artifacts produced and understanding developed.*

**Table 5.4: Product Alienation Coding Scheme**

| Code | Valence | Name | Definition | Example Indicators |
|---|---|---|---|---|
| **PA-1** | A | Cannot Explain Own Work | Student produced a solution but explicitly acknowledges inability to explain it, justify design decisions, or reconstruct reasoning | "I couldn't fully explain why the code works"; "I implemented it but couldn't tell you why"; "the AI wrote it and I'm not sure I could recreate it" |
| **PA-2** | A | Confidence-Capability Gap | Student expresses high confidence on assisted work but acknowledges or predicts inability to perform independently; or retroactively recognizes over-confidence after exam | "I felt good about the assignment but froze on the exam"; "I thought I understood until I had to do it without help"; predicted success but reported failure |
| **PA-3** | A | Ownership Denial | Student explicitly attributes intellectual work to AI rather than themselves; describes final product as "not mine" or "the AI's work" | "the AI figured that out, not me"; "I don't feel like the code is mine"; "it solved the problem for me" |
| **PA-4** | A | Opaque Understanding | Student uses code or applies pattern without being able to articulate underlying principles; understands *what* but not *why* | "I know how to use it but not why it works that way"; "I followed the pattern but couldn't explain the reasoning behind it"; pattern recognition without pattern thinking |

#### 5.3.2.1.2 Process Alienation (PRA)

*Separation from the creative work of problem-solving.*

**Table 5.5: Process Alienation Coding Scheme**

| Code | Valence | Name | Definition |
|---|---|---|---|
| **PRA-1** | A | Prompt-and-Accept Workflow | Student describes workflow as iterating prompts until receiving acceptable output, without critical evaluation of intermediate steps; "shepherding" behavior |
| **PRA-2** | A | Compelled AI Use | Student reports feeling pressured or obligated to use AI despite preferring independent work; competitive pressure; fear of falling behind |
| **PRA-3** | A | Reduced Problem-Solving Agency | Student describes bypassing or outsourcing the problem-solving phase itself — moving from problem statement directly to AI output without intermediate reasoning |
| **PRA-4** | A | Mechanical/ Unsatisfying Experience | Student reports the process feeling mechanical, rote, or intellectually hollow, even when successful; absence of engagement or meaning |
| **PRA-5** | E | Preference for Independent Work | Student explicitly expresses preference for working without AI, or reports greater satisfaction/engagement when working independently |

#### 5.3.2.1.3 Species-Being Alienation (SBA)

*Separation from higher-order cognitive development and metacognitive growth.*

**Table 5.6: Species-Being Alienation Coding Scheme**

| Code | Valence | Name | Definition | Example Indicators |
|---|---|---|---|---|
| **SBA-1** | A | Absent Metacognition | Entry lacks any reflection on the student's own thinking processes; no evidence of self-examination about how they learn, reason, or develop | Surface-level task description only; no "I noticed that I…"; no reflection on cognitive process; entries that describe *what* happened without examining *how* they thought |

| Code | Valence | Name | Definition | Example Indicators |
|---|---|---|---|---|
| **SBA-2** | A | Grade/Efficiency Optimization | Student explicitly prioritizes completion, grades, or speed over understanding; frames AI value primarily in terms of time-saving or grade impact | "it helped me finish faster"; "I used AI to make sure I got full credit"; "I just needed to get it done"; instrumentalizing learning as grade production |
| **SBA-3** | A | Pattern Recognition Without Pattern Thinking | Student can identify or name patterns but cannot explain why a pattern fits a problem, what alternatives exist, or what design principles are at stake | "I recognized it was an Observer pattern"; correct identification with no justification; unable to reason about *why* this pattern vs. others |
| **SBA-4** | A/E | Dependency Awareness | Student explicitly acknowledges reliance on external resources (AI or otherwise) for thinking they recognize they should be doing independently; meta-awareness of intellectual dependency | "I realized I couldn't start without looking something up"; "I depend on AI more than I should"; awareness of the gap between current capability and desired autonomy |
| **SBA-5** | E | Emerging Autonomy | Student demonstrates or explicitly describes developing independent design thinking — reasoning from principles, tolerating ambiguity, making defensible design decisions without external validation | "I figured out why this pattern fits without checking"; "I felt confident defending my choice"; "I could explain the trade-offs myself"; counter-evidence of SBA |
| **SBA-6** | E | Transfer of Understanding | Student describes applying pattern thinking in new contexts, other courses, or novel problems — evidence that understanding has become generative rather than situational | Application to real-world examples; transfer to other classes; novel problem decomposition; counter-evidence of SBA |

#### 5.3.2.1.4 Social Alienation (SA)

*Preferential engagement with AI over peers; reduction of collaborative intellectual life.*

**Table 5.7: Social Alienation Coding Scheme**

| Code | Valence | Name | Definition | Example Indicators |
|---|---|---|---|---|
| **SA-1** | A | AI Preferred Over Peers | Student explicitly states preference for AI assistance over peer collaboration, study groups, or instructor help | "it's easier to just ask AI"; "AI explains things faster than asking someone"; "I don't really work with classmates anymore" |
| **SA-2** | A | Reduced Peer Collaboration | Student reports decreased engagement with peers for intellectual work; AI has functionally | No mention of peer interaction where expected; explicit avoidance of study groups; "I work alone mostly" |

| Code | Valence | Name | Definition | Example Indicators |
|---|---|---|---|---|
| | | | replaced collaborative problem-solving | |
| **SA-3** | A | Avoidance of Intellectual Friction | Student reports avoiding the productive discomfort of encountering competing approaches, peer critique, or alternative problem framings | Preference for AI's convergent responses; avoiding situations where their thinking would be challenged; "AI doesn't make me feel dumb" |
| **SA-4** | E | Peer Engagement | Student reports active peer collaboration, values peer critique, or describes learning from alternative approaches — counter-evidence of social alienation | Study group work; peer review; "my classmate showed me a different way"; counter-evidence of SA |

#### 5.3.2.2 Intensity Scale

**Table 5.8: Intensity Scale**

| Score | Label | Description |
|---|---|---|
| 0 | Absent | No evidence of this indicator |
| 1 | Mild | Weak or ambiguous signal; hedged language; student shows some awareness |
| 2 | Moderate | Clear indicator present; explicit statement or clear behavioral evidence |
| 3 | Severe | Strong, unambiguous indicator; student expresses it without apparent awareness as a problem |

**Examination Analysis.** Exams provided both quantitative performance data and qualitative evidence of understanding. Each exam question received a numerical score based on correctness, but explanation quality was assessed separately. Students could score well on implementation while scoring poorly on explanation, revealing product alienation. For pattern recognition questions, correct identification with weak justification indicated surface pattern matching rather than genuine understanding. For design problems, solutions were evaluated for both technical correctness and architectural reasoning—could students explain why their pattern choices were appropriate, what trade-offs they considered, what alternatives they rejected? Performance patterns across the three exams revealed whether AI conditions affected long-term retention and transfer. Students who performed well on Midterm 1 (after mixed AI exposure) but

poorly on the Final Exam (requiring synthesis) suggested that AI assistance produced temporary capability without durable understanding.

**AI Interaction Log Analysis.** Logs were analyzed for workflow patterns and critical engagement. "Shepherding" behavior appeared when students repeatedly modified prompts until AI produced working code without evaluating intermediate suggestions. "Drifting" appeared when students critically examined AI outputs, identified problems, and made deliberate modifications based on understanding rather than trial-and-error. For Exercise 7's three-phase documentation, the quality of critique revealed whether students possessed independent judgment about code quality. Superficial critiques ("this code looks fine") indicated reliance on AI authority. Deep critiques identifying specific security vulnerabilities or design violations demonstrated autonomous evaluation capability. Time allocation within logs also mattered: students spending most time prompting versus most time evaluating revealed different relationships to the tool.

**Integration Across Data Sources.** Different alienation types required different evidentiary combinations. Product alienation emerged most clearly through the gap between exam performance and journal confidence—students predicting success but demonstrating inability. Process alienation appeared in journal workflow descriptions validated by log behavior—students claiming critical engagement while logs showed shepherding patterns. Species-being alienation required primarily journal evidence of metacognitive depth, with exam performance providing supporting evidence of whether students developed transferable thinking. Social alienation depended on journal reports of collaboration preferences and help-seeking behavior. This mixed-methods approach enabled detection of meta-alienation: cases where

students experienced dependency as capability, reporting high confidence while demonstrating profound separation from independent problem-solving ability.

These instruments and analysis methods transformed the theoretical framework of pedagogical alienation into an empirically investigable phenomenon. By capturing students' independent capability (exams), subjective experience (journals), and actual behavior (logs), the research design made visible the relationships between students and their intellectual work across varying AI conditions. The systematic coding of journals for alienation indicators, combined with examination performance analysis and behavioral evidence from interaction logs, enabled detection of all four alienation types. What remained was to examine whether pedagogical alienation actually manifested in students learning design patterns with AI assistance. Chapter 6 presents the findings from this investigation, revealing patterns that confirm, complicate, and extend the theoretical framework.

## 5.4 Ethical Considerations

This research received approval from the Metropolitan State University Institutional Review Board under expedited review procedures (IRB protocol No. 2357418-2). The research qualified for expedited review as it involved minimal risk to participants and employed standard educational research methods. The proposal and supporting documentation were submitted through IRBNet on August 15, 2025, with approval granted on August 18, 2025—ten days before the Fall 2025 semester began. The approval remains valid through August 18, 2026, covering the full duration of data collection and analysis.

On the first night of class, the research project was described as exploring how the use of GenAI affects the development of pattern thinking. Students were informed that participation was voluntary and would not affect their grades; the instructor would not learn who participated

until after semester completion and grade submission. Students received an informed consent form, which the thesis advisor collected while the instructor was absent from the classroom. The form explained that participants' coursework could be used for research purposes under anonymous coding. Students could withdraw consent at any time by contacting the thesis advisor without the instructor's knowledge.

The dual role of instructor and researcher was managed with care. All course content, whether included in the research or not, was pedagogically valuable, ensuring that all students performed the same tasks regardless of participation status. This meant the instructor remained unaware of who chose to participate. The journal entry prompts and assignment reflections did not ask questions that would elicit information about a student's participation choice.

Preliminary analysis of journal entries began during the semester, but this was anonymized programmatically Student submissions were coded using Python scripts that renamed and organized materials according to an anonymization system. For example, journal entries were exported from the learning management system as CSV files containing student names and responses. Python scripts processed these files by mapping student names to anonymous codes, then exporting responses to files organized in folders named with student codes. Primary analysis of the data began after the semester concluded and all grades were submitted. Paper exams were hand-coded with student identification numbers immediately after collection, creating temporal distance between grading and research analysis.

Data is stored on a password-protected Microsoft Office 365 cloud drive provided by the university. In compliance with university policy, data will be retained for three years, after which electronic files will be permanently deleted and paper exams will be shredded.

# 6 Findings: Evidence of Alienation

Up until this point, we have argued that GenAI constitutes an ontologically new mediation of the learner's relationship to their own learning. It affects not only outcomes and performance but the very nature of learning itself. We have suggested that GenAI can interrupt the development of understanding by obviating the struggle necessary for genuine learning to occur. We have further theorized that GenAI can induce alienation, separating learners from both the products and the processes of their education, and we have described the mechanisms through which this alienation manifests.

In what follows, we turn to the learning of actual students. The question is not whether they used GenAI, nor how much they used it. Rather, we ask whether the four forms of alienation identified by Karl Marx appear in recognizable form when students learn abstract concepts such as design patterns with the aid of GenAI. Here, we test the theory developed in the previous section against the lived experience of students in an advanced computer science classroom.

Because alienation is primarily an experiential phenomenon, qualitative data predominates in these findings. What matters is how students describe their relationship to their work, not merely how they perform on assignments and exams. Journal entries, assignment reflections, surveys, and critical analyses of GenAI were coded using the scheme presented in Chapter 5. Exam data appears where it performs specific theoretical work, highlighting the gap between a student's expressed confidence and their demonstrated independent capability, a gap that itself functions as an indicator of alienation. The analysis draws on all four AI conditions and all four data sources, though not symmetrically. Some findings are most visible in the first

assignment condition, while others do not manifest until the final one. We follow the logic of the argument rather than the chronology of the course.

Each form of alienation is examined in turn. First, we ask whether product alienation was evident: were students able to explain what they submitted, and did they regard it as their own work? Second, we consider whether alienation from the process of production manifested in their engagement: were students meaningfully involved in developing understanding, and could they reproduce the problem-solving activity that led to their submission? Third, we examine whether species-being alienation emerged: were students cultivating their human capacities and developing transferable forms of skill? Finally, we assess whether social alienation took hold: were students cut off from their peers and distanced from the community of learners?

We then return to the research questions introduced in Chapter 1 before turning to a phenomenon that emerged in this study: alienated empowerment.

## 6.1 Baseline Problem-Solving Methods and GenAI Utilization Patterns

The students in ICS372 came to this course with a diverse set of approaches to GenAI integrated learning. In the two journal entries prior to the first AI condition assignment students recounted their past approaches to problem solving, their views on GenAI tools, and how they integrated the tools with their learning. All students had previous experience with GenAI tools such as Google's Gemini and ChatGPT. No student reported a lack of familiarity with GenAI.

In the first two journal entries we saw two distinctive approaches to problem solving: *understanding before action* and *dive in and iterate*. Students described these approaches without explicitly naming them. Neither approach correlated to a specific pattern of GenAI interaction, nor a general view of GenAI and its role in their learning process. Furthermore, neither approach is more valid than the other and both approaches have the potential to yield genuine learning.

The eleven students in the *understanding before action* cluster expressed this approach in variant forms of "I can explain this in my own words." They used this as an internal benchmark that, once achieved, indicated to them that they were ready to develop a solution. S18 articulated this in terms of plain-language narration: "I usually don't begin implementation until I can explain the problem to myself, as if I were describing it to someone else." S14 used the ability to "explain the problem to myself or to someone else without confusion" as a determination of whether or not they were "ready to move forward."

This approach is not simply a preparation technique, but also a claim of cognitive ownership. These students prize understanding over production and consider it a prerequisite for solution development. After the first lecture on the Strategy and Observer design patterns, these students explained a metacognitive shift from "how do I code this?" to "what kind of problem is this?" In Journal 2 (J2) that asked students how the exposure to pattern changed their thinking, the students in this cluster expressed that the patterns were a concept that enable new ways of problem-solving. S24 wrote "Now, before I start, I ask myself if a pattern would help." S15's approach shifted from "jumping straight into coding" to "paus[ing] and ask[ing] myself: 'Is there a pattern here?'" S07 reported that learning patterns would cause them to "pause before coding and ask myself where a pattern fits."

For this group of students, learning design patterns reinforced their existing problem-solving process. They envisioned themselves as continuing to take the time to understand the problem before coding but with the additional step of try to discover the underlying structure of the problem and what patterns it required. These students' approach is potentially vulnerable to GenAI induced alienation. If GenAI can produce a functioning solution without the student

needing to reach their internal benchmarks of understanding they risk bypassing the learning they value.

Six students described their approach as *dive in and iterate* using trial and error as the primary method of producing solutions. S21's cavalier technique of "throwing something together and hoping it works" represents the most extreme form of this method. Other students were more deliberate. S27 describes learning from "trial and error and implementing what I've learned from past mistakes in building on top of my new knowledge." S17 is aware that "usually the first idea in my head doesn't solve the task I'm working on but it gets closer each time."

These students prioritize production over understanding, but they are strategic in applying this method. They understand tradeoffs and the risks—understanding may not develop or working solution could be discovered unintentionally rather than intentionally constructed. The value of this approach comes from the fact that it solves the problem of not knowing where to begin; having a partial or broken solution is a foundation upon which to build understanding through iteration.

After the first design patterns lecture, some students in this cluster experienced a cognitive shift in their approach. S21's report in J2 was the strongest example of this: "Before, I would just dive into code and fix problems as I went. With patterns, I'm starting to think ahead more." The introduction of patterns turned their *code and hope* approach into *plan then execute*. S27 framed design patterns as a "power up" that "helped me learn how to think more systematically." The trial and error approach is susceptible to GenAI induced alienation by way of the first solution generation. If the student uses GenAI to develop the foundation upon which they iterate, they are unlikely to understand why it does not work as intended and therefore the development of genuine learning may be hindered.

In the first two journal entries a couple of themes emerged regarding students' prior experience with GenAI. The primary self-reported interaction pattern involved using GenAI as a bounded tool. Students using this pattern treated GenAI as a supplemental resource in their learning and problem-solving process. The other pattern revealed that a group of students relied too heavily on GenAI to their detriment—experiencing forms of alienated learning prior to this study.

For those that used GenAI as a bounded tool this meant different things. S01's sophisticated use of GenAI reduced its role to that of an intelligent search engine. They claimed that prior to this study they asked only used Gemini to answer conceptual questions and documentation to guide implementation specifics, not to write code on their behalf. To S06 GenAI threatened their capacity for learning if used to generate code they could not understand writing that "as a student, it is important for me to learn the fundamentals and how things interact when coding before I can reliably ask AI to write code for me, or else I would not be able to properly prompt it or dissect issues with the code given." This principled stance, if followed throughout their educational journey, has the potential to guard against GenAI induced alienation.

A separate group of students expressed that GenAI had failed them in various ways. S21's expressed frustration with ChatGPT writing that "a lot of the time it has not been helpful to me because it will try to change my code too much. It often wanted me to use things I haven't learned yet instead of telling me what was exactly not working with my code." This is not simply a complaint about the accuracy or effectiveness of ChatGPT, but a struggle with the role that ChatGPT played in the learning process. It took over the problem-solving process of its own accord when prompted for help, going so far as to implement solutions beyond the student's

understanding. S04 admits that they "can be too lazy and decided to follow anyway until it fooled me." This student is admitting that they used GenAI to bypass the learning process—ceding their judgment to it—only to be fooled.

These early journal entries provide us with a baseline against which to compare student behavior throughout the rest of the study. We can determine whether students maintain their preferred problem-solving methods and GenAI utilization patterns on the assignments through their self-assessments in later journal entries. The following sections will present these findings and the degree to which GenAI induced alienation occurred.

## 6.2 Product Alienation

We see product alienation occurring when a student does not feel that the final solution is their own work. It is evidenced by direct reports to that effect as well as admissions that one is not able to explain their solution to others or that they could not reproduce the solution without the help of GenAI. The indicators of product alienation make it particularly detectable when assessing student narratives of their experiences in the four GenAI conditions.

The first structured AI condition assignment (A1) required students to use GenAI in the implementation of a simulated real-time stock trading dashboard that used the Observer pattern. The students used the GenAI tool of their choice and had the freedom to use it as much or as little as they wanted and in whatever manner seemed best to them. Along with the implementation students submitted an AI interaction log and a reflection on their experience completing the assignment. Journal 3 (J3) came after this assignment and directly asked the students to reflect on their experience using GenAI. Their responses in this entry represent the peak occurrence of product alienation. The entries, coupled with the AI interaction logs and

reflection from the assignment, reveal many instances of the product alienation indicators. This AI condition was the peak for indicators of product alienation.

Every student that submitted a critical analysis with A1 exhibited the Cannot Explain Own Work (PA-1) indicator to some degree. S08 submitted code that did not compile because they were unable to fix the implementation and were even unable to direct the GenAI tool do so for them. S27 reported that they did not even attempt to run the code written by their tool of choice. In their journal entries students were more explicit about their relationship to their submissions. S21 "leaned heavily on AI to even know what files to create and what each file should contain…I also implemented many solutions that I could not explain well to someone else." For S04 the GenAI tool produced code that while it worked in isolation it was "incompatible with my project as a whole," and they felt they "had no choice but to ask the AI to debug its own output repeatedly." S08's entire submission was written by GenAI. They expressed frustration that because of the time limit and the requirement to partner with GenAI that "it would be unrealistic to be able to understand everything to the level of fully explaining it to someone else."

The Confidence-Capability Gap (PA-2) indicator shows up later in the semester after the first midterm exam (E1) when students were unable to rely on external tools. In Journal 5 (J5), S21 stated that they believe themself to be "pretty good at recognizing what the answer should be, but [they] struggle to explain why it belongs there." This student considered themself to have command of the material until the exam when they could detect the correct answer, but could not justify their decision. After midterm 2 (E2) students were given a survey in lieu of a journal entry which asked the about specific questions on E2 and their confidence in how well they understood questions versus the confidence in the correctness of their answers. One of the questions on E2

involved the creation of a sequence diagram. S19 strongly agreed that they understood the question but strongly disagreed that their answer was correct. S08, S04, and S14 experienced the same phenomenon but to a slightly lesser degree.

The Ownership Denial (PA-3) indicator also surfaced in J3 for many students. S01 wrote, without any apparent concern, that "Gemini directly gave me the code used in my final program." S08 says of their A1 submission that "genuinely nothing feels like my own intellectual work." S04 echoed this sentiment saying that after the "back-and-forth with AI, I felt conflicted…It didn't feel like my intellectual work anymore." This student got their program working, but "[b]y the time things finally came together, I couldn't tell what parts of the final product were truly mine and what parts were simply the AI patching its own errors. That left me uneasy, almost like I had lost my sense of authorship." S21 admitted their inability to recreate their solution to A1, and that when they "look at the final code, if feels mostly like AI's work…I still don't feel like I own the solution."

A handful of students wrote things that revealed the Opaque Understanding (PA-4) indicator. S21 is representative here as well writing that they "copied in concurrency features like thread-safe lists and dispatcher queues. They worked, but I would have a hard time teaching someone else why they were needed or how exactly they functioned behind the scenes." S06 explained in their reflection in A1 that they were "not really sure how [they]'d be able to critique" the AI generated code. On Assignment 3 (A3) students were given the option to use GenAI or not, but were required to justify their choice. Here a few students said their decision to use GenAI was prompted by a lack of understanding of the course material. S22 wrote that "I knew AI would be beneficial because I was still gaining familiarity with this factory pattern" and

S09 chose to use GenAI because they "did not find many solutions or examples in the slides or book."

When students used GenAI for their course work they were more likely to experience product alienation. They struggled to explain their own work and to understand the code they submitted. Many reported that they could not reproduce the solution on their own without GenAI's help. Still others saw a contrast between what they thought they were capable of and what they could actually do when GenAI was not available.

## 6.3 Process Alienation

Using GenAI as a replacement for the act of coding has the potential to separate students from the creative work of problem solving and therefore can induce process alienation. Students who make heavy use of GenAI when constructing their assignments may be able to go from an understanding of the problem to the solution with no knowledge of the intermediary steps. They would be unable to justify the decisions made during the coding practice. GenAI induced process alienation overlaps with product alienation with regard to indicators. Often the process alienation is what leads to product alienation since students fail to engage meaningfully with the problem they are trying to solve as they offload the problem-solving to a GenAI tool.

In A1, where students were required to use GenAI, we see evidence of the Prompt-and-Accept Workflow (PRA-1) indicator. This shepherding behavior was the dominant interaction pattern, with 53% of students using this pattern only and 29% using a hybrid approach of shepherding and drifting. The most explicit example of this was students copying the entire assignment specification into the GenAI tool and asking for a complete implementation. S12 "made the file structure…then asked the AI agents exactly the requirements outlined in the assignment." S01's interaction log documents seven interactions, each one concluding with

acceptance of the generated code. In the evaluation sections, they wrote: “This will definitely remain in my code. I see no issues with implementing this,” “This once again seems like a great suggestion,” and “This is typical and standard practice.” This student did not describe any critical assessment of the generated code, nor any indication that they modified the generated code based on their judgments of it.

Another indicator of process alienation is Compelled AI Use. Students experiencing this form of process alienation may not choose to use GenAI if they did not feel external time pressure or peer competition. S08 speaks to this directly in the survey when explaining why they did not answer one of the questions to their satisfaction: “Not having enough time to learn it, because of being bombarded with assignments and feeling the pressure of completing them using AI rather than taking the time to learn.” Ostensibly this student would prefer to take their time to complete their assignments without GenAI, but chose to use it to keep up with their demanding schedule. As a result they did not develop the understanding they had hoped.

Reduced Problem-Solving Agency (PRA-3) also indicates that a student is experiencing process alienation. The students in this situation may want to understand or create the solution on their own, but using GenAI causes them to miss out on this opportunity. Other students may not realize the tradeoff they are making and reach for GenAI to create things they would be capable of doing themself. S23 wrote one long prompt for A1 and submitted the generated code without reviewing it or writing the README themself. S08’s J3 entry exhibited severe reduced agency. They did not understand the assignment as written and asked GenAI to explain it to them. After the explanation they still did not “grasp how each individual class/interface was coded.” S12 and S15 did something similar in that they asked GenAI to explain the assignment to them and write

the code. Understanding the task, the first step of problem-solving, was outsourced to GenAI. These students traded their agency for expediency.

When the work itself becomes unsatisfying or feels mechanical, the student may experience process alienation. The Mechanical/Unsatisfying Experience (PRA-4) indicator showed up multiple times in the classroom. Often students experience this when GenAI is a shortcut to work. S04 appreciated that "AI could produce code quickly," but was disappointed when "instead of leading me toward a deeper understanding, it sometimes layered confusion on top of confusion." Had S04 written the code themself they may not have been so frustrated since they would have been "involved in the step-by-step construction of the solution." S08 used GenAI to write their AI interaction log for A1 because "this approach was so much faster than starting to write the documentation from scratch." On A3, S04 used GenAI "to generate the solution in pieces rather than as a whole." The act of writing and coding here turned into a repetitive input-output interaction with GenAI instead of a creative process.

The final indicator of process alienation is a Preference for Independent Work (PRA-5). Students exhibiting this indicator are not necessarily free of process alienation, but may instead be recovering from or reacting to it. S04's comparison of A1 (AI-required) and A2 (AI-prohibited) is illuminating as they found even though A2 was more time-consuming than A1 it was "far more intellectually engaging…The learning process was active rather than passive." In the AI-usage justification in A3 (AI-optional), S04 explained that even though they knew they would be submitting incomplete work it was worth it for them to "decide not to use AI to assist with the code for this complex problem…not because I am confident, but because I wanted to try." When S21 compares their experiences on A1 and A2 they report that they prefer a mixed

approach, making use of GenAI deliberately in certain parts of their work so they can keep their "attention on design choices rather than spinning on syntax."

It is telling that no indicators of process alienation surfaced when students described their experience of A2, other than saying that they had a preference for this way of working. When GenAI is removed students may struggle to implement and debug their code, but they are more intellectual engaged, curious, and exploratory. S23 "really enjoyed [A2]. Honestly, it was more challenging than [A1], but that's exactly why I liked it." When the option of using GenAI is removed students engage more with the problem-solving process and find the experience of completing their assignments more interesting and rewarding. This counter example elicited when asked to compare their perceptions of working on A1 and A2 is indicative of the alienating experience when working with GenAI. In other words, students self-report enjoying working on an assignment without GenAI versus with GenAI even though there is nothing qualitatively different between the two.

## 6.4 Species-Being Alienation

Humanity's species-being is the capacity for engaging in free, conscious, and creative labor; to make one's own life and the species itself the object of one's activities. In the classroom this translates to the ability to make one's own thinking the object of one's activities and to engaging in conscious and creative work that intentionally deepens one's understanding. GenAI has the potential to induce species-being alienation as it can become the engine of thinking that replaces the cognitive labor of learning, making metacognition neither necessary nor possible as the student performs little thinking upon which to reflect. Species-being alienation occurs on the formative vector of learning. A student is not reflecting on the work performed along the productive vector thereby disrupting the development of genuine learning.

The Absent Metacognition (SBA-1) indicator occurs more frequently early in the course and is evidenced primarily in journal entries. The journals themselves were designed to induce metacognition, so writing that is primarily descriptive rather than reflective belies the presence of SBA-1. S10 is the most severe case of SBA-1. Throughout all thirteen journals they do not once examine their own thinking process—never answering the questions about why they approached a problem a certain way, or how they are developing as a programmer based on what they were learning. GenAI is, according to them, the cause of this gap as they write in J1: "Artificial Intelligence (AI) has become a common part of my academic routine. It is fast, accessible, and capable of producing information on almost any topic." This framing of GenAI without any reflection on how it has affected their learning process, suggests that it has replaced the problem-solving process thereby leaving little thinking to think about. S14's journals are also primarily descriptive. When explaining what they do to complete assignments they show the steps they took—read lecture slides and textbook, discuss with tutors—but do not reflect upon how those steps develop their cognitive capacity. While S15 shifts to deeper reflection in the later parts of the semester, their early journals are similarly descriptive only—what they did, what resources they consulted. We can also see possible evidence of SBA-1 through the lack of journal submissions. S17 only submitted six journal entries total. This student has not made their own learning the object of their attention.

Grade/Efficiency Optimization (SBA-2) occurs when the student decides that learning is secondary to the completion of the coursework. For these students, achieving a certain grade and completing the assignments on time is the primary purpose of their education, not the development of transferable knowledge. At the outset of the semester S12 oriented their learning as an instrument of developing skills with market-value. Though they did, by the end of the

semester, show development along the formative vector, developing understanding and performing well on the final exam. S22 prized action and effort over deep thinking writing in the reflection portion of A2 that they didn't consider alternative approaches than the one they implemented because "The important thing to me is constantly working towards a solution instead of simply waiting around doing nothing." Their exam score trajectory showed declining performance as the semester went on scoring 27/50 on E1, 16/50 on E2, and 46.5/100 on E3, all the while not developing metacognition over the course of their 13 journal entries. S03's "confidence in how I approached the question because I knew I was better equipped with resources," indicates not an internalization of knowledge, but an over-reliance on tools, including GenAI. Their inability to complete A2 and their low scores on E1 and E2 show that the optimization for efficiency with tools led to an underperformance when the tools were unavailable.

We have talked in previous chapters about the ability of expert programmers to see the underlying structures to build a mental model of a solution to a problem. This is pattern thinking and it is a marker of expertise. Though the students in this study are just that, we can see emergent pattern thinking in some of them, while others rely on surface details to detect existing patterns only. This Pattern Recognition Without Pattern Thinking (SBA-3) indicator shows up across the cohort though not uniformly. The students who exhibit this indicator of species-being alienation will be able to identify that a problem calls for a particular design pattern, but will not be able to justify why. In J3, the journal after the AI-required A1, S19 wrote that while they implemented the Observer pattern with the help of GenAI, the code itself and the purpose of the pattern confused them. Without the help of GenAI on A2, they wrote that "The need for it to be flexible and extendable calls for Strategy pattern," naming the properties of an application that

makes use of the Strategy pattern, but failing to explain why this particular problem was best solved the implementation of the Strategy pattern. Their declining exam performance throughout the semester show a lack of internalization of the concepts, as E2 and E3 required code analysis and system design problems with requirements that did not specify the presence or use of a particular pattern. By the time students took E3, they had been exposed to at least 6 patterns in depth and therefore the determination of which pattern was the correct one was a matter of selecting from a variety of possibilities rather than on E1 where it could only have been Observer or Strategy. S09 also exhibited SBA-3 on A1 when upon looking at the GenAI drafted code, they found it expressed confusion about the underlying logic. S17 determined the use of the Strategy pattern on A2 "by looking at the class names." S10 and S14 employed pattern vocabulary at the level of only describing properties, but could not justify the consequences of their design choices.

Dependency Awareness (SBA-4) can indicate either species-being alienation or species-being realization depending on the context. This means that we need to connect the students who experience SBA-4 to the presence or absence of Emerging Autonomy (SBA-5) and Transfer of Understanding (SBA-6) for that same student. Furthermore, students may also not exhibit SBA-4 at all and therefore be experiencing species-being alienation in the form of dependency on GenAI, or they may be aware of the dependency potential of GenAI and have already taken steps to avoid it. Therefore four distinct relationships to SBA-4 need to be analyzed to create the full picture: 1) SBA-4 absent with no SBA-5/6, 2) SBA-4 absent with SBA-5/6, 3) SBA-4 present with later SBA-5/6, and 4) SBA-4 present with no SBA-5/6.

**Table 6.1: Dependency Awareness and Formative Development: Four Patterns Across the Cohort**

| | SBA-5/6 Absent | SBA-5/6 Present |
|---|---|---|
| SBA-4 Absent | Empty Set | S06, S09, S11, S16, S20, S21, S24, S25, S26 |
| SBA-4 Present | S10, S22 | S02, S03, S04, S15, S19, recovery cluster |

In this cohort of students the SBA-4 absent with no SBA-5/6 group is a null set. It appears that the instrument used to gauge species-being alienation, the journals, had an effect on the appearance of this indicator. By asking students to reflect on their use of and relationship to GenAI, we seem to have elicited enough reflection that even students with low metacognitive capacity saw dependency when it existed. This is not to say that all students experienced SBA-4 and documented it.

The SBA-4 absent with SBA-5/6 group comprised S06, S09, S11, S16, S21, S24, S26—and S12, S20, S25 to varying degrees. These students either did not have a prior dependency on GenAI or they took active steps in this course to correct it. The absence of SBA-4 is not a gap in the data; it is a non-event. The metacognitive capabilities of this group of students, combined with the active engagement in the process of learning, protected against the development of a dependency on GenAI. The final exam scores for 5 of these students—S06, S11, S16, S21, S24—were the highest scores in the class. This shows that despite earlier experiences of alienation that some of these students manifested, they were able to counteract them and develop genuine learning. In J1, S26 wrote that they were "aware of the risks of over-relying on [GenAI]…I try to use AI only for brainstorming or getting unstuck, not as my main solution." Prior to the first week of class this student had taken steps to ensure that dependency would not develop. Similarly, S06 sees GenAI "as a supplemental guide, similar to asking a peer for advice." They retain metacognitive control, only directing queries to GenAI when they already have code that needs review or an idea they want vetted. These two students are examples of prospective dependency awareness. They both understand the potential for GenAI to supplant their own thinking, and have structured their use of it accordingly.

There was a smaller group of students who experienced dependency on GenAI early in the semester, became aware of it, and then course corrected. This SBA-4 present with SBA-5/6 recovery cluster—S02, S03, S04, S15, S19—took their dissatisfactory experience working on A1 (though some did not see it that way until after completing A2), and they made decisions to avoid the loss of agency and ownership of their learning. When working on A1, S15 believed "having AI assistance available gave [them] confidence to begin implementing the structure, even when [they weren't] sure how to handle the more complex parts." But after A2, they realized the tradeoffs indicating that the experience was "also a bit detached, almost like I was editing someone else's draft rather than building my own solution from scratch." Although S15 did choose to use GenAI on A3 (AI-optional), they did so with more intention than that had on A1, using GenAI only to "speed-scaffolded a clean factory." After the four AI condition assignments were completed that they had come to "decide whether and how to use AI forc[ing] me to actively assess my own understanding rather than defaulting to external assistance. In those moments, learning felt slower but deeper."

S27 experienced a similar trajectory. After the realization on A1 that without GenAI they would not be able to implement the requirements beyond the basic Observer pattern covered in class. After A4, reflecting in J7 upon all four AI-condition assignments, they wrote that "having the autonomy to choose my level of AI involvement made me feel accountable for the entire workflow from design to execution," and in J9 they reveal that "when AI was prohibited, I was immediately slowed down by having to generate every bit of reasoning for myself. Mistakes and gaps in my knowledge were obvious. That discomfort was productive." This student realized on their own the value of desirable difficulties and intentionally created them for themself, choosing to take responsibility for their learning and showing an elevated level of introspection.

This cluster of students shows that the experience of GenAI induced alienation is not an all-or-nothing condition. They can become aware of their dependency and realize what it costs them. Once they do, they can find ways of learning that create the discomfort that is required to develop deeper understanding. This can include the use of GenAI since, as we have seen here, it is not just that a person uses it, but how they use it that creates an alienated experience.

The final pair of students, S10 and S22, are in the SBA-4 present with no SBA-5/6 set. These two students show the limits of reflection through journaling. While they recognized that they developed a dependency on GenAI, they did not see this as problematic and did not change their behavior as the semester went on. S10 understood that "if [they] cannot explain a solution, then [they] do not truly own it," but then instead of taking the time to understand the code, they went on to implement, with the help of GenAI, the retry and backoff logic that they couldn't fully explain. Later in J9 they write: "I work best as an AI-optional learner: I use AI to set direction, then I take over and do the real design work." While this seems like SBA-5, GenAI is doing the heavy lifting of defining the shape of the program. In J13 they claim that even with GenAI pair coding "the thinking stays mine." On E3, however, S10 scored 35 of a possible 110 points. They did not attempt the bonus question which asked only for a reflection on their learning over the semester. Despite the claims of independent thinking and doing the design work themselves, this knowledge was not internalized and when working without GenAI, S10 was unable to perform adequately.

S22 struggled consistently throughout the semester. In J11 they describe classic pattern matching behavior: "I followed templates from examples, I notice that is a trend for me where I recall work seen from the course lectures and try to see if it applies the same way in a given scenario." On A4 their approach involved wholesale delegation of cognitive effort to the GenAI

writing “AI has extraordinary capabilities, it can generate large amounts of code with no effort… it provided a structure for you.” S22 used GenAI on A3 because they “knew AI would be beneficial because [they were] still gaining familiarity with this factory pattern” and because they did not “know how and when to use the factory pattern.” The assignment was not a learning event for S22. Even though they were aware that they were using GenAI to compensate for the gaps in their knowledge, they prioritized completion over learning. Like S10, they were able to use the language of metacognition in their journals, but their exam performance was near the bottom of the class, indicating a lack of SBA-5 and SBA-6. These two students performed the productive vector of learning, but they failed to run the formative vector and genuine learning did not occur.

Emerging Autonomy (SBA-5) and Transfer of Understanding (SBA-6) are a refutation of species-being alienation. The students who achieved these two indicators found ways to ensure they developed understanding of the material despite using GenAI in their coursework. What these students do shows intentionality and self-awareness. When GenAI “suggested a helper method that [S11] didn’t really understand. Instead of just copying it in, [they] stopped and tested out [their] own version.” S16 developed the capability of holding multiple competing problem-solving approaches in their head and evaluating them on their individual merits. They were also able to reason from the domain requirements of a given problem to the pattern best suited to solving that problem, instead of simply pattern matching. When S20 was struggling to complete A2 they realized they were approaching the problem wrong and “the breakthrough moment came when I decided to treat it like a pipeline.” This shows the ability to recognize and correct one’s own conceptual errors, a signifier of SBA-6, only possible because this student worked through their confusion themself.

The evidence shows that GenAI induced species-being alienation does occur in educational settings, but it is not an inevitability. Students who have sufficient metacognitive abilities and make deliberate decisions to maintain the desirable difficulty, are able to develop genuine understanding.

## 6.5 Social Alienation

Social alienation occurs in a pedagogical context when GenAI replaces the peer collaboration that is constitutive aspect of educational life. Students have different perspectives, approaches, and opinions. Peer interaction provides encounters with authentic cognitive differences, the productive discomfort of being challenged, and the need to articulate and defend one's reasoning. GenAI, on the other hand, is by design reflective of its interlocutor and tends toward sycophantic agreeability. Students stand to lose a lot by replacing human interaction with GenAI.

The nature of the coursework and the fact that the class was taught online reduced opportunities for spontaneous interactions among students. Social alienation is the least well-documented form of alienation in the data. In the first two journal entries some students mention the use of peers as a source of help during the problem-solving process. For the most part no mention of peer collaboration is documented in later journals. This is surprising given that students worked in groups on a collaborative project during the second half of the semester. The presence of social alienation must be inferred from its absence in student narratives and the way that students talked about their use of GenAI.

The only explicit mention of human interactions as a source of aid during the problem-solving process is in the journals of S14. They write that when they are struggling on a problem they cannot solve on their own they "usually turn to the tutoring center and get help or email the

professor with [their] question." This unprompted spontaneous mention of seeking out social interaction suggests a preference for it over interaction with GenAI.

S10 appears here as the only sufficiently well-documented occurrence of GenAI-induced social alienation. They seem to prefer GenAI because "it is fast, accessible, and capable of producing information on almost any topic." Unlike humans which are difficult to connect with and the process of information sharing with them is much slower. We see a possible instance of the AI Preferred Over Peers (SA-1) indicator. The absence of any mention of peer interaction across all of their journals is a possible sign of Reduced Peer Collaboration (SA-2). Avoidance of Intellectual Friction (SA-3) is difficult to code here, but the absence of friction is present in this student's journal as they use GenAI which tends to converge on their own framing rather than challenging or contesting it. The omission of Peer Interaction (SA-4) reveals social alienation.

Other students provide hints of evidence for social alienation, but the prime indicator of social alienation is the lack of evidence of social interaction. This is not sufficient to declare with certainty that GenAI induced social alienation occurred in this cohort of students. We can presume that had social interaction occurred that at least a few students would have documented in one of many narratives the students provided.

## 6.6 Alienated Empowerment—The Phenomenology of Inverted Alienation

During the analysis of the data a phenomenon surfaced that we did not hypothesize: *alienated empowerment*. This is when a person perceives GenAI-mediated dependency as empowerment. Furthermore, this alienation operates on a higher order and masks itself to the subject. Alienated empowerment is a claim about the gap between *felt* and *structural* capability—can you do what you think you can do. Students experiencing it do not see the gap between their self-perception and their actual capabilities until they are tested without the

presence of GenAI. Even then, some individuals experiencing alienated empowerment attribute their underperformance to external factors such as environment, time constraints, or exam format unfairness.

The key evidence for alienated empowerment is exam performance. Exams were structured to mimic the problems from exercises and assignments. They required students to perform the same tasks as they did earlier in the course with access to external resources (GenAI, Internet search), but in miniature and in a domain that was not known ahead of time. Each question on E2 was of the form $Q_n = Q_{n-1} + \Delta_n$, where $\Delta_n$ denotes the marginal design task added at stage $n$ — a structure that reproduced, in examination form, the sequential dependency students had exercised in every AI-assisted exercise and take home assignment. The exam should have been familiar in terms of process, differing only in terms of context—a novel domain to apply the software development process, not a difference in method. The students performed uniformly poorly with a mean score of 52.7%.

The findings reveal a stark contrast between the self-reported pre-exam confidence and the individual exam scores across the entire cohort which indicates that alienated empowerment is not an isolated phenomenon affecting only a handful of students, but a pattern distributed across a substantial portion of the students. Only 18 students completed the survey, but 12 of these students reported feeling “Mostly prepared, but uncertain about some aspects” prior to the exam. One student reported feeling “Completely prepared,” while only two reported feeling “Not really prepared.” The “Mostly prepared” group had scores ranging for 26-100%, which means that feeling “mostly prepared” was not an indicator of exam performance. This variance in performance compared to self-reported preparedness is the statistical signature of alienated empowerment. A student’s subjective readiness decoupled from structural capability means that

self-report carries no predictive power. It is the inflated sense of readiness beyond ability to execute that signifies alienated empowerment.

Three questions from the survey are notably revealing. Q11 aimed to expose meaningful differences in the problem-solving approaches between independent assignments and exams. Q15 asked students to report their pre-exam confidence level. Q17 tried to ascertain how students had developed their design thinking throughout the semester.

**Table 6.2: Selected Post-Midterm 2 Survey Questions**

| Q | Section | Question | Type | Response Options |
|---|---|---|---|---|
| 9 | Design Thinking Mode | When working on the design questions (Q3–Q6), which statement BEST describes your experience? | Multiple Choice | Generative Thinking: reasoning from principles, responsibilities, and design constraints; Pattern Matching: recognizing which pattern this matched and adapting familiar solutions; Procedural Application: following steps or templates taught in class; Uncertain/Stuck: kept starting and stopping, trying different things |
| 11 | Homework vs. Exam | When working on design assignments at home, how do you typically approach design problems? | Multiple Choice | I think through problems similarly to the exam, just with more time; I feel more confident because I can validate my thinking against external resources; I approach design fundamentally differently when I have access to resources; I rely heavily on external resources to scaffold my thinking process; My approach varies depending on the assignment |
| 13 | Homework vs. Exam | How do those resources help you? | Multiple Choice | They help me understand concepts so I can apply them independently; They give me examples I can adapt; They provide templates or structures I can fill in; They generate solutions I can use or modify; They validate that I'm on the right track; I don't use external resources much |
| 15 | Preparation & Learning | Did you feel prepared for this type of exam? | Multiple Choice | Yes, completely prepared; Mostly prepared, but uncertain about some aspects; Somewhat prepared, but significant gaps; Not really prepared |
| 17 | Preparation & Learning | How would you characterize the design | Multiple Choice | I've been developing my own design thinking and making |

| Q | Section | Question | Type | Response Options |
|---|---|---|---|---|
| | | work you've done this semester? *(Be honest — there's no wrong answer.)* | | independent decisions; I've been learning by studying examples and adapting them to new situations; I've been following templates and guidance to complete assignments; I've been relying heavily on external resources (AI, peers, examples) to scaffold my work; My approach varies depending on the assignment difficulty |
| 18 | Meta-Cognition | Beyond "knowledge of Chapters 5–6," what do you think this exam was really testing? | Open Response | — |
| 19 | Meta-Cognition | Complete this sentence: "The hardest part of learning object-oriented design is…" | Open Response | — |
| 20 | Looking Forward | Given your experience on this exam, what specific skills do you need to work on before the final? *(Be specific and actionable.)* | Open Response | — |

**Table 6.3: Preparedness Distribution**

| Preparedness Level | N | Mean E2/50 | Range |
|---|---|---|---|
| Completely prepared | 1 | 74% | — |
| Mostly prepared | 12 | 58.3% | 26–100% |
| Somewhat prepared | 3 | 48.0% | 32–80% |
| Not really prepared | 2 | 43.0% | 28–58% |

It should be noted here that the survey seems to have had an ameliorative effect on some students. The survey revealed to students how their pre-exam confidence contrasted with their E2 performance, and as a result some students were able to modify their behavior sufficiently enough that their final exam scores were significantly higher. S03, S18, and S27, who all self-reported as "Heavy scaffolding," seem to have recognized their gap and disrupted their alienated empowerment. S14 and S19 made a partial recovery. For S22, the survey was not sufficient as an intervention and their performance persisted unchanged. The recovery arc will be discussed in the next chapter.

E2 consisted of three parts: an analysis phase, a design phase, and an implementation phase. The implementation phase consisted of writing on paper Java code that matched the UML class diagrams created in the design phase. The code did not need to be compilable, nor syntactically correct; it only needed to represent the class designs and inter-class communication from the sequence diagram. Under exam conditions, the cohort was not able to reliably complete the third section and as a result it was dropped from the students' score calculation. In the score calculations here we retain section 3 in the point calculation since assessing a student's total awarded points against the total possible is true indicator of their performance, despite the post-hoc grade adjustment. One student (S19) articulated the cascading design as an unfair exam feature in Q18: "It was testing the ability to create an intricate system with no chance to approve the earlier tasks that are required to do the harder tasks."

The "unfair format" critique, however, cannot account for two additional exam features that make the cascading failure more theoretically significant, not less. First, students were permitted to bring a full sheet of letter sized paper—both sides—of self-prepared notes to all exams. No restriction was placed on the content or format of these notes. A well-organized note sheet could contain example diagrams, code snippets, worked examples, process checklists, and notation conventions. The uniform failure on E2 occurred in the presence of these student-prepared notes. This reframes the assessment of what the exam was actually testing—not memory, but the ability to apply a process to a novel domain using whatever summary of that process the student had chosen to record. The note sheet itself is a metacognitive artifact, and its preparation is a diagnostic act. Knowing what to include and what to exclude entails knowing what information is essential and what the student may not be able to recall without an aid. A student who genuinely understood the object-oriented design process we had been covering in

class would have included example UML diagrams, definitions of key terms, and process guidelines among other things. The student who had experienced GenAI-mediated learning, who had delegated the problem-solving and construction process, would not understand which material was essential to carry with them to the exam. They would not know what they did not know, and using GenAI to create the note sheet would be counter-productive as it would not have the necessary context to understand the exam structure or the students knowledge gaps.

The second feature of E2 that lessens the strength of the "unfair format" critique is the extensive in-class pre-midterm 2 preparation. The week before the exam, the instructor reviewed all of the material that had been covered since the first night of class until that point in the semester. Students were told explicitly what types of tasks the exam would involve—including the necessity of creating a sequence diagram based on a detailed use case of their own creation. Despite this explicit forewarning, some students failed to attempt creating a sequence diagram, while others produced diagrams that were not sequence diagrams. Sequence diagrams are the most common diagram used in the course lectures and on slides to visually describe communication among software components. Thus the failure to accurately create these diagrams is not a content-coverage failure—students were told explicitly what the requirements would be, had already created such diagrams on assignments, and had a week to prepare for E2. The failure to translate specific instructions into proper notes and execution on the exam indicates two things, both consistent with alienated empowerment. Either students felt they were sufficiently prepared and did not need to incorporate the instructor's advice, or they did prepare but did not understand what it meant to independently create a sequence diagram, because their prior experience of creating sequence diagrams involved the use of GenAI.

Six students stand out as representative of the alienated empowerment phenomenon. Their self-reported preparedness level, preparation approach, and exam scores reveal the presence of alienated empowerment. The common theme among these students is a heavy reliance on external resources to either create or validate their solutions.

**Table 6.4: Students Exhibiting Alienated Empowerment***

| Student | E2/50 | Q15 | Q11 (Home vs. Exam) | Q17 (Self-Assessment) | Final Arc |
|---|---|---|---|---|---|
| S10 | 58% | Not really prepared | Approach fundamentally differently with resources | Templates/guidance | No recovery (35) |
| S22 | 32% | Somewhat prepared | Validate against external resources | Heavy scaffolding | Persistent (46.5) |
| S17 | 28% | No survey | No survey | No survey | Disengaged (14) |
| S19 | 48% | Mostly prepared | Validate against external resources | Varies | Partial (67) |
| S14 | 40% | Mostly prepared | Same approach, just more time | Varies | Partial (75) |

**S10 is included on structural grounds rather than survey-inflation grounds: his Q15 response was accurate ("Not really prepared"), which distinguishes him from the inflated-confidence signature. He is included because his exam arc, journal corpus, and post-survey trajectory constitute the most theoretically complete alienated empowerment case in the dataset. S17 completed no survey; inclusion is based on the cohort's largest E1→E2 collapse and disengagement pattern.*

The most peculiar case of alienated empowerment is that of S10. On the survey they were clear that they were "Not really prepared" for E2; they do not report the inflated confidence that is a core signature of alienated empowerment. But the rest of their work reveals an interesting picture. On the survey, they honestly reported for Q9 and Q17 a heavy reliance on templates, and on Q11 they revealed that they "approach design fundamentally differently when [they] have access to resources". Their response to Q18 shows that they understood what E2 was testing: "It was meant to test your critical thinking. In the work field you may not have access to external material due to the lack of time you may have to complete the project." S10 can articulate what is wrong with their learning, but this fails to translate into meaningful learning. Throughout the semester, S10 exhibits all four types of alienation to a severe degree. Nevertheless, the narrative

record this student provides shows an active engagement with the reflective activities—completing 13 journals and the survey—and the assignments. They document their GenAI interaction patterns well and discuss patterns by name. They do not report dissatisfaction, confusion, or the sensation of not understanding. The subjective experience is that of a student highly involved with the activities of learning, but learning fails to take place. This is the structure of alienated empowerment—participation that generates the feeling of learning without the structural reality of it.

S22 exhibits alienated empowerment in the form that we would expect to see it—inflated confidence with performance that does not match. What sets S22 apart from the other students is that their alienation persists beyond the intervention of the survey. On Q15 they selected "Somewhat prepared, but significant gaps." This is a marginal deflation of confidence relative to "Mostly prepared," but it seems that it would still warrant performance on E2 greater than a score of 32%. The survey confronted him with the gap but did not dislodge his account of why it exists. Their answers to the other questions reveal why. S22 admits a heavy reliance on external resources for both scaffolding of their work and validating their thinking. But on Q13 they claim that these external resources help them to understand concepts so that they can apply them independently. The failure to apply concepts on the exam shows that there is a disconnect between self-perception and their practice. If S22 truly believes that they are applying concepts independently, then they will not perceive failure on an exam as something they could have avoided by preparing for differently. This exam performance persists until the final where they achieved a score of 46.5%. This disconnect is made sharper when comparing AI interaction logs from assignments against what they reported in the survey. On A1, they began work by completely delegating the work of writing the code to GenAI, and their final reflection is

unambiguous: “Everything I asked before was helpful and was part of the final solution provided by AI.” This is not scaffolding assistance as claimed on the survey, but total productive substitution. The interaction log from A4 shows that they dumped the entire assignment specification in a single prompt. Even though the tool fails to generate complete code—stopping generation mid-if-statement—the response is to re-prompt until a working solution is generated. In the penultimate journal S22 writes: “A lot of my approach with a problem is attempting to recall techniques I have seen before…to actually recognize a design problem and build your own solution is still very difficult for me.” This is not early-semester ignorance, but an admission that in the last week of the semester this student has not internalized the fundamental competency this course aimed to teach.

Two students, S19 and S17, show similar trajectories across the first two exams, both achieving scores of 86% (above the class mean) on E1 before collapsing on E2. E1 was qualitatively different than E2 as it covered a smaller set of material (it occurred in the fifth class session), and primarily tested detection and classification—students had not yet been exposed to the design or implementation phases, and they had only been exposed to two design patterns so they had a fifty percent chance of getting the pattern detection correct. Since E1 rewards pattern detection the AI-scaffolded recognition-level familiarity may have been adequate preparation. However, the sequential design structure of E2 seems to have been beyond their capabilities as it required integrated independent process execution. S17 experienced the greatest drop in score from E1 to E2—a 58 percentage point collapse, which led to a complete withdrawal in engagement. They only completed 6 of the journals, the lowest count of anyone, admitting in J10 that on E2 that they “just honestly gave up once [they] got halfway through.” They did not complete the survey, and so any insight that may have been gained in doing so is lost. The

disengagement continued, and on the final exam S17 only answered the first four questions (short essay) and two later pattern detection questions, but did not receive full credit on any of them. The remaining questions were left blank, including the bonus reflection question. S19 felt "Mostly prepared" going in to E2, but scored 48%. When working on the design specific question they reported they were uncertain about how to proceed, and kept trying different things just hoping that something would work. The template-matching skills cultivated with GenAI did not suffice in helping S19 arrive at solutions. This is consistent with their reliance on external resources to validate their thinking as they report on Q11. The open-ended survey responses reveal that S19 lacks the exact skills that they have offloaded to GenAI. On Q19 they wrote: "The hardest part of learning object-oriented design is to understand how to actually design the project with the new information," and on Q20: "The ability to foresee how my design choice will affect the whole project. The ability to understand the unspecified requirements. The ability to modify my previous decisions." These final competencies—foresight, flexibility, iterative refinement—are precisely what the AI-mediated production has been doing for them. E2 revealed the gap and the survey helped S19 articulated it. As a result they experienced a moderate recovery and scored 67% on E3.

S14's alienated empowerment is milder than that of the other four students, and arises not just from GenAI, but also from the journal substitution effect. It seems that through the completion of the 15 journal entries—the highest in the cohort—they generated a narrative of growing capability that did not correspond to actual performance. In J8 they wrote: "I now feel more confident designing event-driven systems, UI updates, simulations." Yet on E2, the subsequent exam, they only scored 42%. The journals reveal a real growth in confidence, but the structure to support the confidence did not materialize. Though there was mild improvement on

the final exam with a score of 68%. This student shows that reflective writing can substitute, rather than just consolidate, the formative transformation.

Alienated empowerment poses a complicating factor when assessing student progress in a GenAI classroom. The revelation reveals that student self-reports cannot necessarily be trusted as GenAI use can cause an individual to feel more capable and productive than they are in actuality. While there is the potential to disrupt alienated empowerment with interventions that bring the phenomenon to the attention of the subject, awareness alone does not generate a change in behavior.

## 6.7 Research Questions Revisited

The three research questions posed in Chapter 1 probe distinct layers in the educational labor process as theorized in the dual transformation framework. They trace a development arc of increasing complexity: recognition (RQ1) → transfer (RQ2) → synthesis (RQ3). Each represents a cognitive capacity that is more formatively demanding than the previous, more dependent on genuine productive engagement to develop, and more thoroughly undermined by AI substitution.

1. **RQ1**: *How does AI assistance affect students' ability to detect design patterns in unfamiliar code?* This question falls along the **productive vector at the recognition level**. Pattern detection in unfamiliar code requires that the student be able to perform a diagnostic transformation: taking an unknown artifact and decomposing it into recognizable structural elements. If AI has been performing this diagnostic work during exercises, the student's capacity to perform it independently degrades. RQ1 therefore tests whether productive vector disruption has a measurable effect on pattern-recognition capability as a transferable cognitive skill.

2. **RQ2**: *How does AI assistance influence students' capacity to apply learned patterns across different domains?* This question concerns the **formative vector at the transfer level**. Applying a pattern across domains requires that the student have internalized the pattern's structural logic deeply enough to recognize similar problem structures in unfamiliar settings. This is not rote application (repeating an implementation seen before) but genuine transfer (recognizing that a *new* problem has the *same structural features* as a solved one). This is the species-being capacity the course is designed to develop—pattern thinking rather than pattern recognition. RQ2 tests whether AI-mediated productive labor prevents the formative transformation that produces transferable pattern thinking.
3. **RQ3**: *How does AI assistance impact students' skill in synthesizing multiple patterns into elegant solutions?* This question aligns with **both vectors at their highest level of integration**. Pattern synthesis requires pattern thinking at the design level: not merely applying patterns individually but understanding how they interact, when they compose productively, when they conflict, and how to achieve structural economy across a multi-pattern architecture. This is the most demanding formative goal of the course and the most vulnerable to AI mediation. When AI synthesizes multiple patterns into a solution, it performs a cognitive operation that is, for the novice designer, the apex of productive transformation, thereby eliminating the formative transformation that would have made the student capable of performing it.

### 6.7.1 RQ1: How Does AI Assistance Affect Students' Ability to Detect Design Patterns in Unfamiliar Code?

Pattern detection is the ability to read existing code and see within it the structure, including what design patterns are already implemented, what structural logic governs the code, and what architecture is already in place. GenAI is capable of implementing patterns with near

perfection, but it cannot teach a student how to read a pattern in unfamiliar code. When a student has experienced pattern implementation primarily through a GenAI-mediated workflow, they may develop the ability to detect a pattern when they know what they are looking for or when they see structural elements that closely resemble the archetype of that pattern—a function called 'notify,' an interface with 'strategy' in the name—but they will struggle to see the pattern when it is embedded deep in unfamiliar terrain and implemented across multiple software components.

The Observer pattern assignment, A1, is the most direct source of evidence for RQ1. In this assignment students were required to provide an implementation of the Observer pattern that was structurally different than what they had previously seen in the lecture as it required advanced features such as retry of failed notifications, notification filterer, and multi-threading. The student who used GenAI to implement this assignment will have *seen* an Observer implementation in a novel context, while the students who implemented it themselves, using GenAI for supplementary tasks, will have *derived* the Observer pattern's applicability for this domain.

The evidence reveals that students who intentionally limited GenAI usage when working on problems show superior pattern detection to those who delegated freely. In the early journals S06, S11, S16, and S24 articulated explicit AI-limiting strategies before encountering A1. Their exam performance on pattern-detection tasks was consistently strong. S06 uses AI to brainstorm approaches prior to implementation or to get unstuck mid-implemantion, but resisted the tendency to use it as their main solution provider. Furthermore they verified their work "against *more reliable* sources" (emphasis added), implying that they do not see GenAI as a primarily reliable source of validation. S11 did use GenAI to write code, but they did not copy it in to their project as presented. Instead they "broke it down, line by line, until [they] could explain what it

was doing," then wrote their own version. Even though S16 used typical pattern detection language when they said they "could see the pattern right away," they continued with a justification for why the chosen pattern was suitable based on the structural features of the problem. Their final exam score of 95% shows that this pattern-detection transferred to ever more complex and unfamiliar contexts. S24 struggled on A1 because they first attempted to solve it with a polling architecture, but upon implementation they realized that it would miss events. This led them to change the implementation to the Observer pattern. The false start was a productive failure event that created an opportunity for diagnosis and judgment that would not have been possible had they asked GenAI to create the Observer pattern for them from the start. By maintaining agency in the productive transformation, they completed the formative transformation.

Students with impaired pattern-detection capabilities were those who tended towards full delegation of the productive transformation to GenAI. S10, a self-professed heavy user of GenAI, had flat performance across all three exams. When posed with the challenge of detecting patterns when detection required understanding structural features rather than descriptive features they faltered. It seems that despite continued engagement with the material, the formative transformation required for sophisticated pattern detection did not transpire. Similarly, S19 struggled to detect features when descriptive features of the code did not make the pattern obvious. Their exercise reflections indicate surface level thinking, patterns were named but their usefulness was justified by general properties like flexible and extendable rather than the problem structures to which they are best suited. Using GenAI to implement design patterns on exercises where pattern usage was explicit did not transfer to the ability to detect patterns in unfamiliar domains. A GenAI assisted pattern scaffolding spike early on enabled two students,

S19 and S17, to perform well on E1 where pattern detection was tested. However, as we saw in the previous section, this capability did not transfer to comparable performance on E2 which required them to analyze code containing multiple patterns with non-obvious names.

GenAI-assistance directly affects students' ability to detect design patterns in unfamiliar code in a consistent and measurable direction. Students delegating implementation to GenAI develop impaired detection capabilities. The AI-mediated productive transformation removes the diagnostic and analytic practices that develop genuine pattern reading capability. The effect of this is most visible in the E1→E2 performance trajectory, where E1 success depended on surface-level pattern matching, but E2 required structural thinking and pattern implementation.

### 6.7.2 RQ2: How Does AI Assistance Influence Students' Capacity to Apply Learned Patterns Across Different Domains?

The ability to detect a pattern in existing code or to successfully implement a design pattern when given an assignment that explicitly calls for it, does not ensure the ability to select from a library of patterns the correct one for a given problem and implement it. Such cross-domain pattern application requires the student has developed what the theoretical framework calls pattern thinking. This involves not just knowing, for example, that Strategy pattern is useful when the problem involves interchangeable algorithms, but understanding *why* the structural logic of problems with independently variable behaviors demands the Strategy pattern. This understanding of *why* only occurs through struggle of deriving the pattern from the problem's structure, identifying which parts of the code require decoupling, and recognizing the Strategy pattern's specific form of decoupling complies with that structure.

Evidence for RQ2 comes from looking at the four AI-condition assignments. A2 which required independent implementation of the strategy pattern that was similar to, but not identical with, the context students had seen on lectures is particularly revealing. Since this assignment

prohibited GenAI use and forced independent productive transformation, it tested if the formative transformation during A1 had produced transferable understanding. A3 and A4 provided supporting evidence through the different domains and patterns they used.

Students with strong formative transformation in the AI-required condition assignment showed robust cross-domain transfer capability. Curiously, S24 considered using the Template Method pattern, though it had not yet been covered, as a solution for A2, but ultimately rejected this approach as it would have "locked in the algorithmic skeleton, which is the wrong constraint for a system where each pricing strategy needs to be fully independent." This student extracted pre-existing knowledge and was able to rule out competing possibilities by reasoning from the specific structures of the problem. S21 articulated the transfer by way of ownership when they wrote: "Because the design decisions were mine, I felt more confident explaining and defending them." The metacognitive confidence was earned by performing the productive transformation on their own, and enables this student to reason from derived structural principles rather than received pattern implementations.

The students for whom AI-mediated learning dominated showed domain-specific, non-transferable understanding. S03's active and persistent use of GenAI to help them work through problems step-by-step developed only domain-specific knowledge about the patterns they implemented. On E2 the limits of their domain-specific knowledge was tested beyond the familiar domain, and they scored only 26%. S19 could articulate that the Strategy pattern is "flexible and extendable," but these are descriptions of the Strategy pattern, not justification for its use—a lot of patterns are "flexible and extendable." The failure to develop pattern thinking meant that this student's performance peaked on E1, but plummeted on E2 and barely recovered on E3, indicating only partial pattern transfer development, but not higher level pattern thinking.

A3 permitted students to decide whether or not to use GenAI, self-selecting the degree to which they were involved with the productive transformation, and revealing their disposition toward independent pattern application. The students who used GenAI for only lookup or verification produced sophisticated design rationales. S06, S09, and S24 all document specific structural reasoning for why the Factory pattern was the optimal way of handling object creation in the specific domain. The students that delegated core implementation to GenAI were only able to describe *what* the factory pattern does not *why* it is the ideal pattern to apply for this specific object creation problem. S22's behavior on A3 is the RQ2 problem in its most explicit form. They frame AI as a teacher and use it for "gaining familiarity" with the Factory pattern, but the way this is described in the A3 reflection reveals that the understanding only went far enough to explain why this Factory for this class hierarchy, not that the Factory pattern is suitable generally in situations demanding complex runtime decisions about object creation.

The survey showed further evidence for how AI assistance affects a student's capacity for cross-domain pattern application The students who reported little difference in their problem-solving approach on homework assignments versus their approach on the exams had higher incidence of cross-domain knowledge transfer. This suggests that the productive transformation mode on homework was independent enough of GenAI to be useful on exams. The students for whom there existed a great difference between their style on homework and their style on exams struggled with the exams and reported getting stuck. Since they relied on GenAI for the productive transformation on homework, the exam was the site of formative transformation because it was the only place where they performed the productive transformation. At this point, however, it was not a sufficient enough transformation to create transferable understanding.

GenAI-mediated learning negatively impacts a student's ability to develop the capacity for cross-domain pattern application. The mechanism is a disruption of the formative vector due to a lack of student engagement in the productive vector. Relying on GenAI to produce pattern implementations help students develop domain-specific familiarity with patterns, but do not develop genuine transferable learning.

### 6.7.3 RQ3: How Does AI Assistance Impact Students' Skill in Synthesizing Multiple Patterns into Elegant Solutions?

Pattern synthesis is the apex of the productive and formative transformations that this course aims to develop in students. It requires three capabilities that build on each other: first, an understanding of what an individual pattern contributes; second, the ability to determine which patterns required by the underlying structure of the problem; and third, the ability to conceptualize how different patterns interact—how they can be composed, where they conflict, how to structure them to achieve harmony in implementation. This last capability is qualitatively different from the first two. The student who masters the Observer and Decorator patterns in isolation knows how the implementation of the Observer pattern looks and how the implementation of the Decorator pattern looks. They do not, as a result, know how to integrate them into a system where the Observable object is also a Decorator, nor how to avoid the combinatorial explosion inherent in the tension between these pattern, nor how to achieve the same goals with fewer layers of abstraction. It is only through the experience of attempting combinations of patterns—and the productive failure of combining them incorrectly—that one develops pattern synthesis mastery.

Since A4, AI-required with critical analysis, requires an implementation of the Decorator pattern with patterns encountered earlier in the course, it is the most direct source of evidence for RQ3. Not only did this assignment require students to synthesize patterns, but also to evaluate

the implementation provided by GenAI, critique the design, and make improving modifications. This assignment was specifically designed to force RQ3-level engagement even though GenAI provided the initial pattern synthesis. A4 is a structured attempt to preserve the formative transformation when GenAI is performing the productive transformation. The logic being that if students perform diagnostic work in the GenAI's pattern synthesis, they will develop pattern synthesis capability even if they didn't perform the synthesis themselves.

The evidence suggests that this assignment achieved its aims for the students who already possessed sophisticated pattern thinking (RQ2), while it failed to foster such ability for the students who lacked it. S11 documents their critique of the GenAI implementation by specifically questioning the degree of interface coupling between the two patterns and whether the GenAI's proposed synthesis achieves the goal of the assignment, or if it introduces unnecessary indirection. Their perfect score on the E3 confirm that the engagement on A4 produced durable synthesis ability as E3 required multi-pattern synthesis. S16's evaluation compares the GenAI composition against the SOLID principles and assesses whether it maintains the single responsibility principle across all layers of the implementation. This meta-level analysis evidences synthesis capability that can assess not just that the implementation works, but if it is the most structurally appropriate solution and why.

For students that consistently delegated productive transformation tasks to GenAI, A4 did not elicit RQ3-level thinking. These students were not capable of deriving from structure the justification for a design pattern, and therefore could not asses if a two-pattern composition is structurally elegant. S10 did not submit the assignment which is evidence that they either did not understand how to complete the assignment without the help of GenAI or that they did not attempt it. A score of 35% on E3 confirms that they did not develop pattern-synthesis abilities.

S19's critical analysis describes the Decorator in terms of its additive property—"each decorator adds a behavior without changing the existing structure" (a reformulation of the description of the Decorator pattern from the lecture slides)—without examining how the Decorator and Observer patterns interact structurally in the GenAI implementation. They are capable of describing what each pattern does in isolation, but cannot analyze what their composition produces structurally. The fact that this student did not attempt to answer the pattern synthesis questions on E3, reveals that pattern synthesis skills did not develop.

RQ3 introduces an aesthetic dimension that goes beyond just correctness, but into elegance. An elegant multi-pattern synthesis maintains the inherent properties of each pattern, while also achieving structural flexibility and extensibility, using the minimum amount of abstraction, coupling, and complexity. To identify an elegant solution from a range of possible solutions, requires higher order judgment and discrimination. One must not just be able to see that an implementation is elegant, but also that an alternate implementation would be lacking in one aspect or another.

Again S24 stands out among their peers. Through their experience of failure on A1, where they attempted a polling architecture, then refactored to an Observer architecture, they developed an understanding of what works, but also of what doesn't work and why. In their analysis for A4 they can make the elegance judgment in explaining why the Decorator and Observer synthesis is a better means of achieving the propagation requirement than using polling. Without the productive failure experience on A1, this discernment may not have developed. For the students who only receive GenAI implementations, it has no opportunity to develop. S16's evaluation of the design choices not just in terms of pattern implementation, but also adherence to the SOLID principles. This shows a burgeoning philosophy of object-oriented

design that facilitates an analysis of the GenAI's pattern composition that goes deeper than most of their peers. For this student SOLID has become a filter for evaluating a design and providing grounds for structural critique of an implementation approach. On A2, they evaluate contrasting approaches and are able to discern the elegant solution from the inelegant solution, naming what makes the elegant solution elegant and the inelegant solution inelegant.

The journal substitution effect complicates the RQ3 analysis. Because many journal entries throughout the semester required reflection on decisions, they produced metacognitive narration of synthesis reasoning without necessarily producing synthesis capability. Students who engage deeply with journal prompts develop fluency in describing pattern interactions, even if the pattern interactions they are describing are AI-generated. The journal requirement has the potential of producing metacognitive fluency (being able to describe what Observer and Decorator produce together), without producing synthesis capability in students whose productive transformation is AI-mediated. The clearest example of this is S14, whose 15 journal entries represent the highest reflective engagement in the cohort. They provide confident descriptions of multi-pattern interactions in their journals, but their exam performance does not match their vocabulary.

AI-assistance directly impacts students' ability to synthesize multiple patterns into elegant solutions by removing access to two experiences that develop this skill: the experience of failing at a synthesis attempt (and thereby understanding why a particular composition does not work), and the experience of evaluating competing synthetic approaches against structural criteria for elegance. It is only when the productive transformation is the student's own, that there is the opportunity for these experiences. When GenAI disrupts the productive transformation, the formative transformation does not take place and the student does not

internalize the judgment capabilities required for elegant synthesis. Although GenAI solutions may be functionally correct, correctness alone is not an indicator of elegance. Only the students who struggle with the process of synthesis and learn to derive, attempt, and evaluate synthetic approaches are able to develop the comparative structural judgment required to assess elegance independently.

## 6.8 Summary

The evidence shows that GenAI can induce all forms of pedagogical alienation. Product alienation when it produces a student's artifact and they feel no ownership of it, process alienation when it replaces the problem-solving process, species-being alienation when it inhibits the development of genuine learning, and social alienation when it substitutes for peer interaction. It can also cause alienated empowerment—dependency masquerading as competency. But the relationship is not causal—not every student who uses GenAI will experience alienation—and the distribution is not uniform—a student who experiences one form of GenAI induced alienation may not experience the other forms. We also saw that alienation can be interrupted by interventions during the education journey, for example through reflective journaling that prompts metacognitive development.

# 7 Discussion

To one degree or another, many of us have a suspicion that using GenAI can lead to dependence on it. In some of the studies we referenced in Chapter 2, students were quoted expressing concern that overuse of GenAI would lead to over-reliance. Such concerns are common when human society experiences a significant technical shift. From early concerns in *Phaedrus* that writing would erode memory, to mid-twentieth-century critiques of television as cognitively diminishing, to contemporary debates about smartphones and attention, periods of technological change have reliably generated widely held assumptions about cognitive decline that often outpace the available evidence. What is needed is not a judgment about GenAI but a precise account of what it does and to whom and under what conditions.

## 7.1 Alienated Empowerment as a Theoretical Contribution

Alienated empowerment is more than just false confidence. It describes a phenomenon where the subject experiences themself as capable of performing the labor that GenAI is doing because they are participating—though in a diminished capacity—in the process. They confuse direction setting with authorship, review of output with comprehension, and implementation speed with mastery. A person possessing only false confidence may misperceive the quality of the work—assessing it to be of a higher quality than it is—but they own their participation in the process and they can recreate it. Alienated empowerment affects the perception of competency not at the level of final product—a GenAI created product is often of high quality—but at the level of process.

The dual transformation framework provides a lens which enables us to detect and name alienated empowerment. By separating the practice of education into two distinct but dependent vectors, we can see where in the process GenAI has an effect. Disruptions along the productive

transformation vector can have disruptive effects on the formative transformation vector—either weakening the formative process, or inhibiting it altogether. Alienated empowerment happens when disruption materializes on both vectors and the subject is unaware. This threatens the subject's capacity for learning because they lose the ability to see the gaps in their knowledge. The GenAI tool seamlessly fills the gaps when the subject is working from concept to completion.

Alienated empowerment puts a name to a suspicion about the effects of GenAI on learners and workers. The evidence we presented in the previous chapter shows that it exists and the mechanisms that bring it about. We also saw that it is possible to disrupt alienated empowerment and restore agency to an individual. But this requires sustained effort and the development of metacognitive abilities. Students with higher pre-existing metacognition were less likely to experience alienated empowerment than those who struggled to explain their thought processes.

Previous studies of the effects of GenAI in computer science education focused, for the most part, on outcomes—did students perform better or worse when using GenAI, were they more productive, did the code that GenAI write work? Here we are looking at the phenomenology of education with AI-assistance—how does a student perceive themselves in relation to their work when they use GenAI, were they more or less engaged as a learner when using GenAI, did they develop pattern thinking abilities? We ask not just does a learner learn, but how do they experience their learning and how do they feel about it.

This opens a new line of inquiry for understanding the role of GenAI in education and beyond. We seek to comprehend not just if GenAI leads to higher levels of productivity, but what the consequences of using it are for the individuals who rely on it as a replacement for their

cognitive labor. The contribution we make here is a framework for contextualizing GenAI's role with respect to human activity. Alienated empowerment is not limited to educational contexts and may prove to be widespread phenomenon when explored in different domains. This study is the first to name it.

## 7.2 The Phenomenological Substitution: Theoretical Extensions

Beyond what we presented in Chapter 6, there are some important theoretical points that warrant further discussion. The findings that back these up are not widespread enough within the cohort to count as constitutive evidence, but they are valuable observations that may lead to further investigations. These phenomena provide us with further insight into students' self-perception and behavior with regard to GenAI.

### 7.2.1 Language as Phenomenological Diagnostic

The final question on the final exam (worth five bonus points) was the last opportunity to elicit a reflective narrative from students. The theme of the question was learning and independence. One part of the question asked students to describe their problem-solving process with and without GenAI and their degree of independent capability. The answers to the questions are typical of what we saw in the journals, but some interesting linguistic characteristics reveal more than what the students may have intended.

One student, S22, wrote of their experience learning the strategy pattern: "I can recognize it. I used AI to understand and evaluate it, it had been a great learning experience is it can explain itself very thoroughly." The choice of words belies a particular conception of GenAI. For this student, GenAI is a reflective entity capable of articulating its own thoughts and perspectives. They did not say that "it explained the strategy pattern," they said "it explained *itself*". This is not just an anthropomorphism, it is an unconscious assignment of agency to a

computer program. To S22, GenAI has become an actor in its own right and they are a passive consumer of its thought. Indeed, this student confessed that "implementing patterns are (sic) very hard without AI." The language choices here show a delegation of their agency, and the S22's failure to perform well on any of the three exams corroborates this.

Another student admitted to heavy use of GenAI, and their learning suffered as a result. S14 wrote on the final: "I was primarily trying to direct AI to a solution I had in mind opposed to trying to understand the solutions AI generated…I wanted to see if my work was correct enough for AI to *understand*. Most of the time, AI would fix my solutions" (emphasis added). The choice of *understand* is telling. It presumes that AI has a humanlike pre-existing mental model which it deploys to derive meaning from its input. This perception led S14 to use GenAI as the arbiter of the correctness of their code—if it can *understand* the code, then it must be correct. Still, their code was often not correct and needed repair. This student fared poorly on E1 and E2 with only a slight recovery on E3.

This propensity to use agency assigning language when describing interactions with GenAI was not exclusive to underperforming students. S01 scored 100% on E3, but wrote in their final reflection that they "would usually ask it for what kind of solution IT had in mind" (emphasis theirs). The feature to consider in this quote is the phrase "had in mind." This also positions GenAI as an independent actor with its own perspective and thought process, as an agent capable of having personal preferences or making conscious judgments.

Language and word choices matter because they do not only reveal an individual's thoughts and beliefs, but also because they reinforce the same. How a person talks about their interactions with GenAI may indicate how much agency or independence they are willing to transfer to the agent.

### 7.2.2 The Normative-Descriptive Gap

A charitable view of students experiencing alienated empowerment is that they do not know what a "good programmer" looks like—what it means to be independently competent. The students in this cohort however were, for the most part, capable of articulating this well even if they could not themselves replicate this competence. Another aspect of the final question on the exam was a hypothetical posed to the students: "If you were hiring a programmer, what's on thing from this course they should be able to do without AI assistance? Why is that the one thing that matters?" This question forces a student to evaluate competency without using themself as a frame of reference. A student's answer to this question, contrasted with their exam performance and other self-reports illuminates the gap between their understanding of the normative standard and their failure in their descriptive self-assessment to apply that standard to themselves.

S19 articulates a nearly perfect normative standard for the outcome of the course when answering the above question: "The one thing everybody should be able to do without AI assistance is to understand how design decisions work together…I don't mean pattern recognition, but deeper understanding." Yet when describing their experience answering the design questions on E2, they indicated that they were not sure how to approach the questions and kept trying things hoping that something would work. This student can clearly articulate the desired outcome of this course, and yet remains far from their stated ideal. This is alienated empowerment from a different angle. Awareness of the ideal should drive one closer to it. The final question on E3 required the synthesis of design patterns. S19 left it blank.

Another example of this gap is S14, who answers the above question by writing: "A programmer should be able to understand and implement different design patterns. This is the one thing that matters (IMO) because design patterns are what makes (sic) your code clean and efficient." This articulation of the importance of design patterns did not seem to motivate S14.

They scored less than 50% on both E1 and E2, and 68% on E3. They reported on the survey that their approach to the design questions on E2 was primarily pattern matching. Then when describing the specific skills they need to work on before the final they misdiagnose the problem: "I need to totally change my studying, I need to do example problems and figure out how to work faster and do less time thinking and just put all of my thoughts on the exam paper." Their problem is not that they work too slowly or spend too much time thinking, but rather a diminished capacity for independent execution, or in their words the ability "to understand and implement different design patterns."

What these students highlight is the persistence of a perception gap indicative of alienated empowerment, but signified through a different mechanism. Both of these students were able to accurately describe the normative standard of a competent programmer and why the traits possessed by this programmer are desirable. Yet throughout the semester they made heavy use of GenAI, performed poorly in the absence of external resources, and, after reflecting on why they struggled on E2, failed to align their behavior with their ideal.

### 7.2.3 Intervention Timing and the Consolidation Window

After a catastrophic collapse in scores from the first midterm to the second midterm, five students—S15, S18, S23, S26, S27—who showed early-semester signs of formative disruption and alienated empowerment, went on to score exceptionally better on the final. Two events coincided with the recovery that may have contributed. The first is the survey which was given after E2 to help the researcher and the students understand what the troubles were that led to poorer performance. The second was class session where the instructor worked through E2 live, one problem at a time. The combination of the disruption of the collapse in score on E2, the reflection activity of the survey, and the experience of seeing an expert developer working

through design problems, challenged this cluster of students to re-evaluate their self-perception and modify their behavior.

These five students had developed a mental model of software design as a linear process where one takes a design problem and applies the correct answer. The live design session showed them that design is an iterative and self-reinforcing process. Their J10 entries after the lecture revealed the corrective effect this had on their thinking. S15: “strong design work is not linear or polished, it is iterative, exploratory, and full of small course corrections.” S23: “design isn’t linear at all.” S26: “I assumed that ‘good’ design mean coming up with the right pattern quickly…I thought the process was supposed to feel smooth and linear.” S18: “design thinking is less about guessing the right thing and more about reasoning why a certain structure makes sense.” S27: “I had a more ‘linear’ view: recognize the pattern, draw the structure, implement cleanly.”

This moment of insight makes sense for students who relied on GenAI for help when they got stuck. The interaction pattern with AI is linear: prompt → answer. Users do not see the cognitive effort that goes into the output. GenAI, of course, performs no cognition, but it is trained on innumerable instances of good software design that required significant cognitive effort to create. GenAI conceals this effort, and the students inferred that the design process was as linear as their interactions with it.

The fact that the live-design lecture was a moment of insight for these students suggests the presence of social alienation. Students with disconfirmation avoidance look to GenAI—a disconfirmation-avoidance machine—for solutions as opposed to peers. When a student’s design approach is wrong, GenAI will not tell them their thinking is deficient, it just produces the result of a better approach. This leaves the student with the correct answer, but not the map from their

incomplete conception to the elegant solution. The productive transformation completes; the formative transformation did not begin. But students who collaborate with another and discuss different design approaches, might come to see the shortcomings in their way of thinking and adjust accordingly.

The live-design lecture provided peer friction and a dialectic moment. The recovery cluster had a thesis that design was a linear process, a matter of applying the "right pattern quickly" to a problem. The instructor provided an antithesis: "design work is not linear or polished, it is iterative, exploratory, and full of small course corrections." When the students watched an expert developer work through a design problem from an entirely different premise—policy before pattern, invariants before structure—their thesis appeared to them as the wrong model, and from there formed a new synthesis and a more refined approach to design.

Had the live design lecture occurred before the collapse on E2 that prompted it, students may not have had the same moments of insight. Their struggle to apply what they were taught to a design problem in miniature is what helped them see that their approach lacked crucial components. The failure on E2 may have been painful, but it turned out to be productive for this cluster.

S10 provides a counter-example to these students. They had the same moment of insight during the live-design lecture, but failed to capitalize on the insight and make meaningful behavioral change. In J10 they summarize their takeaway from the session: "A clear model makes pattern choice obvious. A muddy model stays muddy no matter which pattern you apply." The student's insight here that design reasoning starts from first principles (a clear model), did not translate to better performance. Their late semester journal entries show sophisticated design thinking, and they report that they were more deliberate about their design process, less

dependent on AI, and more comfortable with ambiguity. On the final exam, however, they struggled to execute independently scoring only 35%. Many of their answers were incorrect, but in a way that indicates a misunderstanding of the design principle behind the question than a lack of recall. They wrote a lot, but reasoned incorrectly.

The students discussed here show that timely interventions can lead to conceptual shifts. The five students in the recovery cluster were able to make use of their insights by making behavior changes that turned into durable independent design thinking as showcased on their final exam performance. For the student who could not recover, the window for meaningful change may have already closed by the time the insight was gained. They could deploy critical design language and show capacity for self-reflection, but when tested they still came up short.

## 7.3 The Anti-Alienated Empowerment Profile as Pedagogical Target

There are four students in the course who maintained intellectual independence throughout the semester, developed independent design capabilities, and showed no signs of alienated empowerment. These students started the semester with self-discipline and pre-existing rules about how they integrated GenAI into their practice. S11, S16, S21, and S24 share a few common traits that may help guarding against GenAI induced alienated empowerment.

The first trait common among them is a defined standard of what they consider understanding. For these students an assignment is not a task they trade for a grade. It is an event that can produce genuine knowledge. They hold themselves to this standard rigorously. S11 writes that they “don’t use code [they] can’t explain.” This is not a qualified statement. Regardless of code’s origin—their own work, copied from the Internet, generated by AI—if they do not understand how it works they will not use it until they do. GenAI code is not precluded by S11, this is not a refusal to use external tools. Rather it is an internal test of learning that S11

uses to ensure that they continue to engage in the productive transformation because they understand—explicitly or intuitively—that it is required for the formative transformation to take place.

These students embrace failure as information, not as a motivation to outsource cognitive labor. When their attempts to solve a problem do not work as expected they work, not just to fix it, but to understand why. For them a failed design is just as valuable, if not more so than a functioning one. S24 failed to implement the strategy pattern correctly on A2; they coded themself into a corner realizing that "putting the base fare logic directly inside the engine…made the code harder to extend." The thinking process here is not a trial-and-error until something works approach. S24 saw why their approach would not work and modified the design to fit their requirements.

This group also values ownership of their work for its own sake. They maintain participation in the productive transformation because they see it is intrinsically meaningful, not just instrumentally so. They are careful to avoid offloading their cognitive labor to GenAI. S21 expresses this after working on A3 (AI-optional): "because the design decisions were mine, I felt more confident explaining and defending them." This student intentionally chose to complete this assignment without GenAI because they wanted to own the process, not only the product—identifying the productive transformation as the source of intellectual ownership.

Two of these students—S11 and S24—chose not to use GenAI on A3, while S16 and S21 did. The use of GenAI on its own provides no correlation to the development of the anti-alienated empowerment profile. Rather, it is the intentionality and self-discipline that determines if that use is additive or detrimental.

## 7.4 Implications for Pedagogical Design

When educators consider how to integrate GenAI into the classroom, we must take care to ensure that we are still providing our students with the experience they deserve, with the opportunity to develop transferable knowledge and genuine understanding. As we've seen, GenAI does pose a threat to this as it can interrupt both the productive and formative vectors of learning. The four students we just discussed provide a template to build from when considering how to create curriculum that maintains pedagogical integrity without prohibiting the use of GenAI.

We have seen that the experience of alienation is not uniform across students who use GenAI. One student may feel product alienation, but not species-being alienation, while another may only experience process alienation. We may see a student experience product alienation on one assignment, but social alienation on another. The framework here provides a way to understand the student experience when attempting to learn alongside GenAI. We have shown the vectors along which the alienation occurs and we have shown ways that we can detect it. There is also evidence that alienation patterns can be disrupted and students experiencing one of the forms of alienation or alienated empowerment can be made aware of their deficiencies and coaxed into re-engagement.

In STEM classes it is uncommon to spend time writing prose, let alone engaging in metacognitive discussion about your own learning. The journals and reflections on the assignments that students in this cohort performed, gave us valuable insight into whether or not the students are actually learning and to what degree. These narrations also caused some students to pivot in their approach throughout the semester after realizing that they had become reliant on GenAI and that their individual process had not prepared them for the exams. Since GenAI is

particularly skilled in subjects where this is a correct answer (i.e. STEM), educators may want to consider integrating reflection activities on their syllabi. While the ameliorative effect of these activities remains to be seen, the process of reflection will, at the very least, help us to understand how our students' thinking is being shaped by the course material and GenAI.

Additionally, educators ought not prohibit GenAI use altogether. There are many AI-cheating-detection tools available—which themselves are AI—but they have just as much chance of flagging a false positive as they do letting a false negative slide. Our goal should be to help students learn how to preserve their intellectual agency in a world where the ability to offload cognitive labor is a few keystrokes away. The students in this cohort show that it is the relationship that an individual student has to GenAI that can be detrimental to students, not the use of the tool itself.

What one student wrote in their final journal entry shows that our students understand this:

> "I considered myself using AI in a way where, if everything collapsed one day, if the data centers shut down, if Google disappeared, if Stack Overflow stopped existing, I would still be able to build something meaningful because the core design skills live in me, not in the tools. Through this course, I've gained confidence in that part of myself: the part that understands the fundamentals of programming languages, good architecture, abstraction, and the principles that make software actually work. These are things I could apply, teach, and build upon even if all external resources vanished."

## 7.5 Limitations and Threats to Validity

This study took place in a single semester university computer science course where the researcher was also the course instructor. Though steps were taken to anonymize student data there is always the possibility of unconscious bias to emerge and for subtle patterns across student submissions to reveal identity. Furthermore, the researcher was careful throughout the semester to not instruct the students in a specific way so as to influence inadvertently their use or non-use of GenAI, their method of using it, or the perceptions of it generally. Nevertheless, as the researcher was also the instructor and because the course assignments were designed to elicit specific types of responses, students may have been unintentionally influenced. The students were aware of what was being studied and that the instructor would be the one analyzing the data. This may have shaped the nature of their output or the depth of their reflections. The journal entries themselves were not graded on any criteria other than completion, and this was communicated to the students in the hopes that would not feel coerced into answering questions in a specific way.

The journaling process may have influenced students to be more thoughtful about their approaches to the coursework. Typical computer science courses do not ask students to explain their thought process, analyze how their thinking led to a specific conclusion, or divulge their feelings about using a software tool. The journal prompts generated self-reflection at a level that students may not normally apply to their problem-solving process. The requirement to consistently interrogate one's own process could have induced metacognition in students who were not predisposed to it, which may have caused these students to be more deliberate about their GenAI use.

At the outset the class had 31 students, but by the end of the semester only 27 students remained. This cohort constitutes a small sample that limits the generalizability of our findings. No claim is made here that alienated empowerment—nor any other form of alienation that we described—affects a specific percentage of computer science students broadly, or that the patterns documented in ICS 372 Fall 2025 would replicate in another institution, with a different instructor, or in a different course. What the sample size does permit, however, is theoretical generalizability. The alienation claims are grounded in evidentiary depth sufficient enough to warrant further exploration, but not enough to make population-level claims. The 27 students in this class produced deep sources of data—13 journals, a post-midterm survey, four assignments with varied AI conditions and student narratives, and three exams. This depth of data partially compensates for the lack of breadth.

## 7.6 Future Directions

We established the existence of pedagogical alienation and named a hitherto unnamed form of alienation: alienated empowerment. This occurred in a computer science classroom where correctness is tractable. It remains to be seen if GenAI induced pedagogical alienation can be detected—or if it occurs—in domains outside of STEM where it is not simple to make quantitative determinations from exam scores. This study also raises the possibility of using pedagogical instruments to counteract alienation in students. As this was not our focus we cannot make any recommendations. Further work remains to determine which interventions are most effective and to what degree.

The theoretical framework we developed adapted Marx's concept of alienation for educational settings. Another area for research would be to adapt the framework to a modern GenAI-mediated workplace and investigate how workers experience themselves and their jobs

when using GenAI. Many workplaces are beginning to mandate GenAI usage for its employees—especially for workers in the technology space—and the questions raised in this thesis will become all the more pressing for understanding the experiences of knowledge workers in the years to come.

This thesis also raises epistemological, ontological, ethical, and aesthetic questions that are outside its purview. But these questions deserve exploration. Does it matter if a student cannot do something on their own if they know how to direct an AI agent to do it for them? What knowledge can an individual claim as their own when some if it is acquired dynamically from GenAI during the completion of a task? What do educators owe to their students in a GenAI-saturated world—preparation for working in a GenAI-workplace, or the development of transferable knowledge? If GenAI writes the code, and only GenAI reads the code, does it matter if the code is well-designed? We need frameworks to properly contextualize these questions and provide direction to educators, students, and workers.

# 8 Conclusion

This thesis showed that GenAI induced pedagogical alienation is a real phenomenon and that it is measurable. We can detect when a student is experiencing it and upon which transformation vector it occurs. This alienation does not manifest as failure. Indeed, a student who receives consistently high grades may be experiencing deep alienation, while a student who struggles to master the material may be the most present and engaged student in the classroom. These forms of alienation reveal the mechanism behind the effects that GenAI has on a student's ability to detect patterns in unfamiliar code, their capacity to apply learned patterns across different domains, and their skill in synthesizing multiple patterns into elegant solutions.

The dual transformation framework we described shows us where in the education process alienation occurs. Learning takes place when a person makes a productive transformation (raw lumber into a chair, disparate thoughts into a synthesis), and thereby experiences a formative transformation (transferable skill, deeper understanding). The forms of alienation are correlated to the transformation vector. Product and process alienation can occur when there are disruptions on the productive vector—a student receiving code from GenAI that they did not write, or a student using GenAI to offload certain portions of an assignment. Process, species-being, and social alienation can manifest along the formative vector.

Because programming is a craft, students need to develop tacit knowledge through sustained, intentional, and reflective practice. The craftsperson knows how to use their tools to shape material; they are not shaped by them. The carpenter who relies on a jig for every cut never develops the eye to cut freehand. The programmer who relies on GenAI for every implementation never develops the judgment to see when the implementation is wrong. The tool

becomes a prosthetic for an ability the student was meant to develop, and the prosthetic, over time, atrophies the limb.

The three research questions posed in Chapter 1 were not posed only to assess outcomes. The questions isolate a design thinking development hierarchy that maps onto the alienation types in order of depth. The dual transformation framework enables us to see alienation as a structure rather than a set of separate findings—product alienation on the productive vector, species-being alienation on the formative vector. The core components of the hierarchy are pattern detection, pattern application, and pattern synthesis. Persistent product alienation cuts a student off from developing the capacity to detect patterns in unfamiliar code. Without the experience of implementing design patterns, the student will not be able to see the pattern made plain unless they know what to look for. A student who cannot detect patterns cannot then learn to apply patterns. This is species-being alienation made visible. Cross-domain transfer is an expression of higher-order human species thinking, and when programming students fail to develop it as a result of GenAI-induced process alienation, something human has been interrupted. For a programmer, the inability to develop the apex skill of elegant pattern synthesis is a deep failure. Developing the aesthetic discrimination necessary to create elegant solutions requires hours upon hours of writing and rewriting smelly code. Unless used with care, GenAI disrupts learning on both vectors, induces alienation, and inhibits the development of durable pattern thinking.

But so what? If AI coding agents are writing the coding for only themselves to read—if the human in the loop is only prompting, running, and testing—does it matter if programmers develop pattern thinking? Programming is not the craft of writing code—it is the management of complexity. When a programmer short-circuits their pattern thinking capabilities, they shackle

themselves to the present. Pattern thinking unlocks systems thinking. Systems thinking unlocks the imagination. The coding agent only operates on what has been—the systems thinker imagines what could be.

The four forms of alienation *feel* like alienation to the subject, but alienated empowerment is invisible to the student experiencing it. A student alienated in this way may enjoy watching their code run, but they will never experience that momentous rush of insight when, after struggling unproductively for hours, the mental picture of the system snaps into focus and the solution crystalizes. The student thinks they have experienced this flash of insight. They use GenAI as their cognitive engine, assuming they understand what it produces because they told it what to produce. This student cannot perceive the gaps in their knowledge until something shines a light through the gaps. We know now that alienated empowerment exists—not in every student, but some. And we have seen that it can be interrupted, guarded against, reversed.

When the student of today sits at their computer—an open text editor, the white cursor blinking on a black background, an empty buffer revealing what they do not know, the clock counting the seconds until the assignment is due—they have a choice. Do they stare into the void of the buffer, creating programs in their mind until the patterns emerge, and their fingers turn thoughts into a stream electrons that instruct the computer to bend to their will? Or do they switch to a different window and write a prompt? We have seen both students, we know what each gains and what each loses when they make their decision. What they decide the instructor is unlikely to know. The student reaches for their keyboard with the mind that codes.

## References

[1] K. Marx, *The Economic and Philosophic Manuscripts of 1844 and the Communist Manifesto*. Amherst: Prometheus, 1988.
[2] F. Detienne, "Design Strategies and Knowledge in Object-Oriented Programming: Effects of Experience," *Human-Comp. Interaction*, vol. 10, no. 2, pp. 129–169, June 1995, doi: 10.1207/s15327051hci1002&3_1.
[3] A. Robins, J. Rountree, and N. Rountree, "Learning and Teaching Programming: A Review and Discussion," *Computer Science Education*, vol. 13, no. 2, pp. 137–172, June 2003, doi: 10.1076/csed.13.2.137.14200.
[4] E. Soloway and K. Ehrlich, "Empirical studies of programming knowledge," *IEEE Transactions on software engineering*, vol. 10, no. 5, pp. 595–609, 1984.
[5] E. Gamma, Ed., *Design patterns: Elements of reusable object-oriented software*, 39. printing. Boston, Mass. Munich: Addison-Wesley, 2011.
[6] O. Astrachan, G. Mitchener, G. Berry, and L. Cox, "Design patterns: An essential component of CS curricula," in *Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education*, Mar. 1998, pp. 153–160, doi: 10.1145/273133.273182.
[7] E. Wallingford, "Toward a first course based on object-oriented patterns," presented at the SIGCSE '96, 1996.
[8] V. K. Proulx, "Programming patterns and design patterns in the introductory computer science course," presented at the SIGCSE, 2000.
[9] T. Eigler, F. Huber, and G. Hagel, "Tool-Based Software Engineering Education for Software Design Patterns and Software Architecture Patterns - a Systematic Literature Review," in *Proceedings of the 5th European Conference on Software Engineering Education*, June 2023, pp. 153–161, doi: 10.1145/3593663.3593670.
[10] Z. Jeremic, J. Jovanovic, and D. Gasevic, "Semantically-Enabled Project-Based Collaborative Learning of Software Patterns," in *2009 Ninth IEEE International Conference on Advanced Learning Technologies*, July 2009, pp. 569–571, doi: 10.1109/ICALT.2009.106.
[11] D. S. Da Cruz Silva, M. Schots, and L. Duboc, "Fostering Design Patterns Education: An Exemplar Inspired in the Angry Birds Game Franchise," in *Proceedings of the XVIII Brazilian Symposium on Software Quality*, Oct. 2019, pp. 168–177, doi: 10.1145/3364641.3364660.
[12] A. R. L. Loyola Alvarez and S. E. Cieza-Mostacero, "Serious Video Game to Improve the Learning of Software Design Patterns," *TEM Journal*, pp. 2557–2567, Aug. 2024, doi: 10.18421/TEM133-81.
[13] Z. Jeremic, J. Jovanovic, and D. Gasevic, "An Environment for Project-Based Collaborative Learning of Software Design Patterns," *International Journal of Engineering Education*, vol. 27, no. 1, pp. 41–51, 2011.
[14] M. Gladwell, *Outliers: The story of success*, 1st, Back Bay paperb. ed. New York, NY Boston MA London: Back Bay Books Little, Brown and Company, 2011.
[15] K. A. Ericsson, R. T. Krampe, and C. Tesch-Romer, "The Role of Deliberate Practice in the Acquisition of Expert Performance," *Psychological Review*, vol. 100, no. 3, pp. 363–406, 1993.
[16] K. A. Ericsson and K. W. Harwell, "Deliberate Practice and Proposed Limits on the Effects of Practice on the Acquisition of Expert Performance: Why the Original Definition Matters and Recommendations for Future Research," *Front. Psychol.*, vol. 10, p. 2396, Oct. 2019, doi: 10.3389/fpsyg.2019.02396.

[17] S. Baltes and S. Diehl, "Towards a theory of software development expertise," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Oct. 2018, pp. 187–200, doi: 10.1145/3236024.3236061.
[18] P. S. Fasco, "Cognitive Architecture for Adaptive Problem-Solving and Computational Models of Expert Knowledge Acquisition in Computer Science Education," *International Journal of Multidisciplinary Research and Growth Evaluation*, vol. 6, no. 3, pp. 948–959, May–June 2025.
[19] S. Fitzgerald *et al.*, "Debugging: Finding, fixing and flailing, a multi-institutional study of novice debuggers," *Computer Science Education*, vol. 18, no. 2, pp. 93–116, June 2008, doi: 10.1080/08993400802114508.
[20] M. McCracken *et al.*, "Report by the ITiCSE 2001 Working Group on Assessment of Programming Skills of First-year CS Students," *Working group reports from ITiCSE on Innovation and technology in computer science education*, pp. 125–180, 2001.
[21] D. Loksa, A. J. Ko, W. Jernigan, A. Oleson, C. J. Mendez, and M. M. Burnett, "Programming, Problem Solving, and Self-Awareness: Effects of Explicit Guidance," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, May 2016, pp. 1449–1461, doi: 10.1145/2858036.2858252.
[22] T. Lowe, "Debugging: The key to unlocking the mind of a novice programmer?" in *2019 IEEE Frontiers in Education Conference (FIE)*, Oct. 2019, pp. 1–9, doi: 10.1109/FIE43999.2019.9028699.
[23] J. Metcalfe and A. P. Shimamura, Eds., *Metacognition: Knowing about knowing*, 1. MIT Press paperback ed. Cambridge, Mass.: MIT Press, 1994.
[24] E. L. Bjork and R. A. Bjork, "Making Things Hard on Yourself, But in a Good Way: Creating Desirable Difficulties to Enhance Learning," *Psychology and the real world: Essays illustrating fundamental contributions to society*, vol. 2, no. 59–68, pp. 56–64, 2011.
[25] M. Kapur, "Productive Failure," *Cognition and Instruction*, vol. 26, no. 3, pp. 379–424, July 2008, doi: 10.1080/07370000802212669.
[26] M. Kapur, "Examining Productive Failure, Productive Success, Unproductive Failure, and Unproductive Success in Learning," *Educational Psychologist*, vol. 51, no. 2, pp. 289–299, Apr. 2016, doi: 10.1080/00461520.2016.1155457.
[27] R. A. Schmidt and R. A. Bjork, "New conceptualizations of practice: Common principles in three paradigms suggest new concepts for training," *Psychological science*, vol. 3, no. 4, pp. 207–218, 1992.
[28] C. Baek, T. Tate, and M. Warschauer, "'ChatGPT seems too good to be true': College students' use and perceptions of generative AI," *Computers and Education: Artificial Intelligence*, vol. 7, p. 100294, Dec. 2024, doi: 10.1016/j.caeai.2024.100294.
[29] C. J. S. F. Clarke and A. Konak, "The Impact of AI Use in Programming Courses on Critical Thinking Skills," *Journal of Cybersecurity Education, Research and Practice*, vol. 2025, no. 1, Apr. 2025, doi: 10.62915/2472-2707.1220.
[30] M. Lepp and J. Kaimre, "Does generative AI help in learning programming: Students' perceptions, reported use and relation to performance," *Computers in Human Behavior Reports*, vol. 18, p. 100642, May 2025, doi: 10.1016/j.chbr.2025.100642.
[31] M. I. H. Shihab, C. Hundhausen, A. Tariq, S. Haque, Y. Qiao, and B. W. Mulanda, "The Effects of GitHub Copilot on Computing Students' Programming Effectiveness, Efficiency, and Processes in Brownfield Coding Tasks," in *Proceedings of the 2025 ACM Conference on*

*International Computing Education Research V.1*, Aug. 2025, pp. 407–420, doi: 10.1145/3702652.3744219.
[32] W. Takerngsaksiri, C. Warusavitarne, C. Yaacoub, M. H. K. Hou, and C. Tantithamthavorn, "Students' Perspective on AI Code Completion: Benefits and Challenges," May 31, 2024. http://arxiv.org/abs/2311.00177 (accessed Sept. 17, 2025).
[33] F. Nizamudeen, L. Gatti, N. Bouali, and F. Ahmed, "Investigating the Impact of Code Generation Tools (ChatGPT & Github CoPilot) on Programming Education:" in *Proceedings of the 16th International Conference on Computer Supported Education*, 2024, pp. 221–229, doi: 10.5220/0012628000003693.
[34] J. Prather *et al.*, "'It's Weird That it Knows What I Want': Usability and Interactions with Copilot for Novice Programmers," *ACM Trans. Comput.-Hum. Interact.*, vol. 31, no. 1, pp. 1–31, Feb. 2024, doi: 10.1145/3617367.
[35] S. Rojas-Galeano, "New Kid in the Classroom: Exploring Student Perceptions of AI Coding Assistants," Sept. 16, 2025. http://arxiv.org/abs/2507.22900 (accessed Oct. 26, 2025).
[36] W. Gereluk, "Alienation in Education: A Marxian Re-Definition," *McGill Journal of Education/Revue des sciences de l'éducation de McGill*, vol. 9, no. 1, 1974.
[37] K. Kenklies, "Alienation: The foundation of transformative education," *Journal of Philosophy of Education*, vol. 56, no. 4, pp. 577–592, Nov. 2022, doi: 10.1111/1467-9752.12703.
[38] Y.-L. Wong, "Student Alienation in Higher Education Under Neoliberalism and Global Capitalism: A Case of Community College Students' Instrumentalism in Hong Kong," *Community College Review*, vol. 50, no. 1, pp. 96–116, Jan. 2022, doi: 10.1177/00915521211047680.
[39] X. Tan and Y. Li, "The Plight of Alienation in College Students' Learning: An Analysis Based on User-Generated Content," *J. educ. theory pract.*, vol. 2, no. 3, July 2025, doi: 10.62177/jetp.v2i3.453.
[40] D. R. Ford, "From 'Authentic' to Actual Marxist Educational Theory: Advancing Revolutionary Pedagogies," *International Critical Thought*, vol. 13, no. 4, pp. 506–524, Oct. 2023, doi: 10.1080/21598282.2023.2280898.
[41] S. Karayaman, "The Alienating Effect of Technology: Does Technological Innovation Cause Work Alienation," *TR1010*, p. 6, June 2024, doi: 10.26677/TR1010.2024.1418.
[42] J. Wang and W. Fan, "The effect of ChatGPT on students' learning performance, learning perception, and higher-order thinking: Insights from a meta-analysis," *Humanit Soc Sci Commun*, vol. 12, no. 1, p. 621, May 2025, doi: 10.1057/s41599-025-04787-y.
[43] P. McLaren, *Life in schools: An introduction to critical pedagogy in the foundations of education*, Sixth edition. London New York: Routledge, 2016.
[44] P. Freire, D. P. Macedo, and I. Shor, *Pedagogy of the oppressed*, 50th anniversary edition. New York: Bloomsbury Academic, 2018.
[45] J. Prather *et al.*, "Interactions with Prompt Problems: A New Way to Teach Programming with Large Language Models," Jan. 19, 2024. http://arxiv.org/abs/2401.10759 (accessed Sept. 10, 2025).
[46] M. W. Apple, *Knowledge, Power, and Education: The Selected Works of Michael W. Apple*. Hoboken: Taylor and Francis, 2012.
[47] E. J. Mayhew and E. Patitsas, "Critical Pedagogy in Practice in the Computing Classroom," in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, Mar. 2023, pp. 1076–1082, doi: 10.1145/3545945.3569840.

[48] C. Babbage, *On the Economy of Machinery and Manufactures*, 1st ed. Cambridge University Press, 2010.
[49] K. Marx, P. Reitter, P. North, W. Brown, and W. C. Roberts, *Capital: Critique of political economy*. Princeton (N.J.): Princeton University press, 2024.
[50] C. W. Mills, *White collar: The American middle classes*, 50th anniversary ed. New York: Oxford university press, 2002.
[51] H. Braverman, *Labor and monopoly capital: The degradation of work in the twentieth century*, 25. anniversary ed. New York, NY: Monthly Review Press, 1998.