The Stained Glass of Knowledge: On Understanding Novice Mental Models of Computing

Briana Christina Bettin
Michigan Technological University, bcbettin@mtu.edu

THE STAINED GLASS OF KNOWLEDGE:
ON UNDERSTANDING NOVICE MENTAL MODELS OF COMPUTING

By

Briana Christina Bettin

A Dissertation

Submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

In Computer Science

Michigan Technological University

2020

This dissertation has been approved in partial fulfillment of the requirements for the Degree of DOCTOR OF PHILOSOPHY in Computer Science.

Department of Computer Science

DISSERTATION ADVISOR:	*Dr. Linda Ott*
COMMITTEE MEMBER:	*Dr. Charles Wallace*
COMMITTEE MEMBER:	*Dr. Stefka Hristova*
COMMITTEE MEMBER:	*Dr. Robert Pastel*
DEPARTMENT CHAIR:	*Dr. Linda Ott*

Dedicated....

To every role model, supporter, and experience
which has inspired, encouraged, and shaped me into who I am today.

You have invigorated my creativity,
And fueled my passion to positively reshape our world.

And to you, dear reader.
In hopes that this research may in some small way
Motivate you toward reshaping our world as well.

Contents

Acknowledgements

I would first like to extend a wealth of gratitude to my adviser, Dr. Linda Ott. Her guidance has helped me to navigate my growth as an academic, and her support has been unwavering in my research pursuits. I am beyond grateful not only for her allowing me to pursue my interests as a researcher, but for her believing in the value of those pursuits. I hope my efforts and the work presented here have made her proud.

I am also grateful to my committee, Dr. Robert Pastel, Dr. Charles Wallace, and Dr. Stefka Hristova. Their support, input, and guidance have aided me immensely on this journey.

I must of course extend immense appreciation toward my wonderful husband, Nathan. He has supported me each step of the way and believed in me at times when I wasn't sure I believed in myself. He has listened to all of my research difficulties and puzzlings. He has cared for and loved me infinitely. He has never let me back down from following my passion and using it to make positive change in the world. I could not have completed this dissertation, or for that matter, likely much of this doctoral process, without the love and care he has provided me. I love you always, Mr.Squishy — time is nothing.

My deepest appreciation extends to my loving parents, Chris and Tina Bettin. Without their love, encouragement, guidance, and support since my birth, I would not be the woman I am today. They have always believed in my dreams and supported my pursuit of them. They are my fiercest advocates, and I am beyond proud to be their daughter. I will never cease to be grateful that I was blessed enough to have them as my parents. Mom and Dad: I love you to the moon and back — and I will love you forever and like you for always.

Thank you as well to all of my extended family for their love, encouragement, and support. I am lucky to have you all in my life.

A final thank you the close friends and mentors who have supported me on this journey — you have added sprinkles of sweetness all along the way.

Abstract

Learning to program can be a novel experience. The rigidity of programming can be at odds with beginning programmer's existing perceptions, and the concepts can feel entirely unfamiliar. These observations motivated this research, which explores two major questions: *What factors influence how novices learn programming?* and *How can analogy by more appropriately leveraged in programming education?*

This dissertation investigates the factors influencing novice programming through multiple methods. The CS1 classroom is observed as a "whole system", with consideration to the factors present in it that can influence the learning process. Learning's cognitive processes are elaborated to ground exploration into specifically learning programming. This includes extensive literature review spanning multiple disciplines. This allows positioning to guide the investigation. The literature survey also contributes to greater understanding of learning cognition within computing education research through its disciplinary depth.

The focus on analogy with the second question is motivated through the factors observed in the first question. Analogy's role in cognition and in education is observed, and the analogical inclinations of technology as a field are showcased. Stigma surrounds the use of analogy in computer science education in spite of these indications. This motivated investigation on how the use of analogy could be better addressed in programming education in order to utilize its value. This research presents a tool for the design of well-formed analogy in programming to answer this question. It also investigates additional forms analogy can take in the classroom setting, proposing relevant cultural forms such as memes can be analogical vehicles that promote learner engagement.

This research presents a strong case for the value of analogy use in the CS1 classroom, and provides a tool to facilitate the design of well-formed analogies. In identifying ways to better leverage analogy in the programming classroom, presenting this research will hopefully contribute to dispelling analogy's "bad reputation" in computing education.

By exploring factors that contribute to the learning process in CS1, this research frames education design as experience design. This motivates methods and considerations from user experience design, and investigates aspects of the "whole system" that can promote or deter a learner's experience.

This dissertation presents findings on understanding the learner's experience in the programming classroom, and how analogy can be used to benefit their learning process.

Chapter 0

Transparency: Prefacing Context

To truly and honestly investigate novice mental models of computer science, one must acknowledge unequivocally the elephant in the classroom. The tremendous shadow cast over almost all work and discussion of interventions in CS is the egregious slant of the discipline with regards to diversity.

Computer science classrooms have a diversity problem. This problem however does not lie with the individuals, but with the field and educational system itself. If we take a critical pedagogy perspective, we must ask: what is the world like, and why is it like that? At the time of writing, the technological climate is unwelcoming to oppressed groups - in both presentation and in practice. Colloquialisms such as “white dude tech bros” reinforce the idea that technology is for the white and male (and often, the white and male with enough privilege to gain startup funding, which adds wealth and status to the mix). Designed algorithms internalize the prejudices of their creators - passing off these prejudices as mathematical truth to a majority that lacks the tools for critical inference. These programmed prejudices deepen oppression as technology’s adoption expands. Programmer decisions in algorithmic design increase incarceration rates of black individuals, proliferate sexist stereotypes, ignore the imperative of situational context, and can even cause death, destitution, and destruction - at much higher rates for those in oppressed groups.

One may argue that such injustices should cause the oppressed to revolutionize technology. This argument misses the barriers to entry that oppression places. Our field is wrought with sloganism of “anyone can code” as a platitude of diverse welcoming. Indeed, anyone CAN code - but ability is not the sum of the diversity problem. Programming languages such as Scratch and Snap!, developed with the intent to ease entry obstacles to computer science, are plagued with the reputation of being “kiddie languages”, pressing on the perspective of those who engage with them. Such perspective oppression is further found in social stereotypes from the cultural sphere: reminding the oppressed “what a programmer looks like”, despite any sloganing to the contrary. Internet resources, consistently referred to as the way to become a “self-made coder”, best favor the enrichment of those who are able to critically question the resource and develop a dialogue of analysis with found resources. Such criticality may not always be found in the oppressed due to limiting factors of their context, such as time and resources. Educational institutions may even further limit where they should empower, if learners are fostered with a response-regurgitation model that inhibits critical investigation.

Such arguments also miss the mythicalization of technology that has produced a largely technologically illiterate but technologically reliant majority. Programming and technology have been glamorized as high-paying and futuristic, enveloping them in a mythos that it’s okay to “just not get those computer things”. Oppression plays on this myth and wields technology as a weapon: to incarcerate, to segregate, to colonize, and to destroy. The computer is an infallible black-box, which is much too complicated to understand - but yet, “anyone can code”. The contradiction implicit: while everyone CAN code, if everyone did code, the truth would be revealed. Thus, an air of mythos and glamour surrounds technology. Jobs which “revolutionize technology” are cast with long hours and personified by seemingly single-passioned workers. The message? “You need to be a certain type of person to work in tech.” - that person being one which dissuades the oppressed.

These statements in no way mean that oppressed and diverse groups cannot evoke change in technology, or that there are not initiatives and revolutionaries doing honest work toward this goal. Instead, it is laying bare the oppressive, mythicized climate in computer science that presently exists. Globally, there are teachers and revolutionaries working to make changes, and globally, there are students of and diverse contexts, learners oppressed by technology and societies that rise up to use this tool for positive transformation. There is hope, and that hope continues to grow through such critical dialogue, investigation, and praxis.

The laying bare of this climate instead names the implicit problem in any critical investigation and reflection of novices in computer science. If computer science as a climate dissuades the oppressed, models found in the classroom may not be able to accurately reflect their truths, or their truths may be drowned out in data noise.

In this work, I want to acknowledge candidly the limitations of my perspective, and of the perspectives within the data collected. As an investigator, I make my best efforts to share the narratives of Subjects in a truthful frame, and to consider the perspectives that may not be presented. I also do my best to acknowledge my own perspectives and biases, and hopefully, not to allow my frame to repress that of the Subjects. This work cannot ever be considered complete - there is simply no way to present a ’truth’ of novice mental models, because models are a consistent internal dialogue with the world around them. As we teach, that dialogue expands, and that dialogue is consistently switched in differing contexts based on their unique backgrounds.

So what can this work do, if not be complete? It can take a critical perspective of observations, discussions, reflections, and data, and it can use these to inform approaches and ideas. These approaches and ideas, when applied as part of a critical learning dialogue, may have the ability to aid novice understanding. The critically observed perspectives, observations, discussions, reflections, and data may generate new knowledge and bridge understandings regarding the Subjects described and their view of programming. In this understanding, one may be better able to understand and engage in dialogue with their own learners. Tools and approaches utilized to engage with Subjects may find utility, or encourage new thoughts and praxis for instructors.

“Any pedagogy which does not question the status quo tacitly or actively endorses it” [46]. The work in this dissertation was conducted at a university in the midwest region of the United States, within a department with a primary enrollment demographic of white males, matching the university. The city the university is located in is primarily white with a poverty rate of approximately 38%, located within Ojibwa (Chippewa) homelands. The rate of college degree attainment of those living within the city is over 55% for a Bachelor’s degree and above. The university’s tuition is often listed among the highest public institutional tuition for its state. English is the primary language of the university (matching the city at 86% English usage), and those who attend have an expectation of English competency for their studies and work. Over 58% of residents were born within the state, with over 87% being born somewhere in the United States. Of those in the population not from the United States, Asia is the primary origin of birth or citizenship. [134] Active efforts are conducted by many faculty and staff within this department and university to increase enrollment of and opportunities for marginalized groups.

The author attended this university for her undergraduate and doctoral work, attending another Midwestern university virtually for her masters while working in the technology industry as a consultant. The author grew up in the rural Midwest within Menominee (Sioux) homelands and attended a small, predominantly white school within a city with a poverty rate of approximately 28% for her K-12 schooling. The rate of college degree attainment from the author's hometown is less than 10% for a Bachelor's degree, and less than 1% for a graduate degree, with over half of the population having high school graduate as highest level attained. Over 98% of the population speak English. Further, over 85% of people in the author's hometown were also born in the same state with 100% being born somewhere in the United States, suggesting non-nomadic tendencies of the population. The population's growth rate is also in steady decline. [135] The author is white, female, privileged to have not experienced the financial hardship of poverty, and has parents who have both attended college, with a mother who has also achieved graduate degrees.

The author readily acknowledges that the contents of this dissertation, including her perspectives and data, are derived from largely white, English-speaking areas within the midwest United States. It would be remiss to not recognize this status quo's presence within the data and perspectives presented. With critical analysis and reflection, the author hopes that this research can still provide value and insights of value beyond this context, and that encourage continued conversations and research interrogating this status quo and engaging additional groups further.

Chapter 1

Gazing: Introduction

"Starting a new journey may not be so hard; or maybe it has already begun." [38]

Are there learners who "just get" introductory programming, versus those who dont? Instructors often observe learners who seem to take to programming ideas quickly, while others appear to stumble over what seem to be basic concepts. Some learners fly through the design of a chain of if-statements with ease, while others puzzle over how to express the conditional correctly. The word "for" being used to signify a loop might feel like second nature to a learner with prior programming experience, but baffle the completely new programmer. Where object-oriented ideas might illuminate the world of programming for some, others may feel stuck in the weeds of new concepts and syntax.

Every learner has a unique background that positions their learning journey. This prior knowledge is much like stained glass: in places clear enough to easily see new concepts and their connections, but also tinted and cut—refracting new ideas in sometimes unanticipated ways. No learner comes to our classroom perfectly singular, cut, and clear. Each stained glass fragment of knowledge is unique, connecting together to form the mosaic of our learner's world: their mental models.

The supposed, inherent capabilities observed in some novice programmers may come from pre-existing mental models. Such prior knowledge can connect the learner's worldview in a way that readily maps to programming ideas. The learner recognizes programming as analogous to their prior models, which promotes the connection of these entities. This can allow mental models of programming to form more easily from the onset. Ease in connecting prior models of the world may give the appearance of latent intuition. While some learners may develop connections more quickly, others may require more effort to form models of core concepts and thus begin to recognize and develop connections.

If this hypothesis is correct, the key for educators is finding methods to promote development of connections within and across mental models. This can be difficult, as learners have varied backgrounds, confidence, and perceptions. However, it appears worthwhile to investigate how we can foster such connections across all types of learners. The classroom will never fully dismantle the learner's mosaic of prior knowledge and models, nor should it aspire to. This would shamefully lose the diverse perspectives of learners, colonizing their thoughts rather than expanding their view. Understanding the unique mosaics our learners bring in order to meaningfully add to them is valuable for computer science education. Understanding aspects of the unique mosaic a learner brings may be the key to facilitate the learner "getting it".

This dissertation is my exploration into the ways in which novices "make sense" of programming. To "know thy user" is a tenet of user experience design: an interdisciplinary field which motivated my approach. To "know thy learner" is imperative to educational design. Exploring how novices "make sense" of programming requires diving into how we explore the world, how we reason, and how cultural knowledge and personal connection influences that reasoning. This research relies heavily on interdisciplinary methods and literature as a result. In this dissertation, I aim to better understand the novice learner, showcase how this understanding may impact educational design. In understanding novice programmers, I focus on analogy as a tool to connect a learner's knowledge fragments. My interest in analogy grew from my background in user experience.

1.1 Guiding Insight from User Experience

User experience provides valuable insights for understanding the classroom as a "whole system" that affects the learner, as well as for the value of analogy in learning. Centered around understanding users and their goals, user experience (UX) bridges the gap between a person's current state and their goals, while recognizing their capabilities and motivators [155]. The "user" is a person interacting with some system or process to achieve a goal: such as a learner interacting with a class in order to achieve a degree. User experience not only meets needs, but aims to exceed them by eliciting joy and evoking simplicity [124]. It is a field connecting many disciplines, including psychology, programming, design, and education.

The "user experience honeycomb" [118] describes six core tenets to achieving "value" in what you provide. These tenets are:

- **Useful.** What is provided should successfully fill a need—even if it is a need users may not realize they have yet.
- **Usable.** Interactions should be logical for users and reduce difficulty wherever possible.
- **Desirable.** The system or product should be something that the user either recognizes a need for, or comes to want. Usefulness ensures a need *is* fulfilled, desirableness indicates users want *this* system or product to fulfill that need.
- **Findable.** Features and information within the system should be easy and logical to locate.
- **Accessible.** We should design for all possible users, including those who may have physical limitations or other conditions that reduce their ability to access or interact.
- **Credible.** Design should promote user trust in the system, so that guidance and information is believed.

Providing value is key in industrial applications, but it is also key in the classroom. The classroom exists to provide learners with value through their experiences with the curricula.

User experience is typically considered with regard to human-computer interaction, but UX is part of the much broader system of experience design. Users may engage with a physical system or process, requiring considerations from areas such as architectural and civil design. We interact with experiential design each day as we navigate spaces and systems in our physical world.

User experience designers must recognize the notions or conceptual understandings a user likely brings to their system. This allows them to appropriately design a system or process users can effectively engage with. Experiences that facilitate existing knowledge reduce what users must learn. Appropriate scaffolding in novel situations reduces frustration, accelerating understanding. These central properties of user experience design also hold true in the space of good educational design. After all, education *is* an experience we engage with!

Affordance and Signifiers

Affordances and signifiers [126] can begin helping us understand how people recognize actions to take within a system. Affordances are described as our understanding—based on what we perceive—about what purposes a system or entity may serve. Signifiers, meanwhile, are cues and representations that can promote or downplay affordances. This distinction can cause obstacles in our discipline's landscape. The physical capacity of a computer mouse button to be pressed allows action to be done: it has the affordance "is-for" clicking. The visible traits of the mouse showing its potential to depress, such as the buttons and their raised nature, are signifiers. However, everything that appears clickable on our screen is also a signifier! The entire screen has the affordance of mapping the input device's action to visible screen output. This also means that the *entire screen* has the affordance of clicking. The acts of placing buttons, links, and more are the placement of signifers, not affordances. Affordance, in essence, is perceived potential. Our perception of potential comes from how we map, associate, and interpret information.

Mapping within interfaces takes many forms, such as skeuomorphism and iconography. These are representations using glyphs and images that should evoke some sense of meaning in the digital realm. Mapping as a process is entrenched in analogy—a comparison of things in order to better understand or explain them. Comparison of clicking a digital button to physically depressing a physical button caused the pointer finger cursor to become a signifier that maps the action to its real world analog. Reasoning about appropriate analogical mapping is at the heart of good experiential design. Signifiers can provide cues or clues to promote reasoning that shifts an understanding of potential.

When learners code, they are engaging with a purely digital landscape in the representations on the screen. This consideration may help us better understand some of the perceived novelty of programming as a discipline. Where disciplines within the analog realm may have more physical affordances, much of what our learners interact with is through signifiers on the screen. This poses unique considerations for the ways we foster learning in novice programmers. We must recognize perception of the digital space and its potential will limit or inform the actions they take. This is at the heart of user experience design—understanding the user in order to promote interactions with the system that achieves goals effectively. The goal of a programming classroom is of course to teach programming. To best achieve this goal, we must be aware of the signifiers in our classroom spaces, tools, and techniques—in both the digital and analog realms.

1.1.1 Shaping the Experience of Educational Design

Education design is experience design, meaning computer science education can be viewed through a user experience lens. I observed the following correlations between the two:

- Computer Science $\longrightarrow$ Product
- Department, University, Industry, Guardians $\longrightarrow$ Stakeholders
- Students $\longrightarrow$ Audience/Users
- Good Grades, Work/Life Skills $\longrightarrow$ User Goals
- Persistence, Mastery $\longrightarrow$ Stakeholder Goals
- Classroom Components $\longrightarrow$ System "Interface"

Identifying these correlations posed learning design as a problem for which experience design principles and ideas may help. This mapping should of course be considered purely as an exploratory topic for research themes — taking it too far risks making a commodity of or oversimplifying the learning environment. Exploring this mapping to identify possible research themes, however, aided my identification of a concept present in both user experience and education research: mental models. Finding this common ground motivated better understanding student mental models as the focus of my work.

One way that existing mental models are leveraged in new contexts is through **mapping**. As we have seen, mapping is fundamental to a well-designed experience. When a user engages with an unknown new system or process, mapping can provide signifiers, allowing the user to better understand how to navigate. Mapping requires the connection of something known to something unknown—and in order to know what the user understands, we must reason about their mental model.

1.2 Analogy in CS Education

I explore several factors surrounding novices learning to program, with emphasis toward the use of analogy in learning. Despite analogy being commonly used in other fields, computer science education researchers often argue against the use of analogy, especially at introductory levels. This fear of using analogy is often a fear of inappropriate mappings developing, or analogy being unable to encapsulate computer science's novelty as a discipline. This research will identify how imperative analogy is to our cognition. It details the arguments against analogy within computer science education, and works to address the concerns about teaching with analogy. Further, multiple modalities for analogy will be presented.

As our exploration unfolds, I will make the case for the value of analogy as a teaching mechanism. There is no reason to keep this value from our novice learners out of fear, when we can work as educators to address the concerns surrounding the use of analogy. In addition to extensive literature exploration to provide evidence for the value of analogy, this dissertation also contributes:

1. Presentation of a design tool created to promote development of meaningful and well-formed teaching analogies
2. Consideration for the impact of "concrete analogical representations" that use physical presentations and systems to ground programming concepts
3. Evidence for the use of Internet memes from popular culture as an analogical reasoning tool
4. Reflections from higher level students on their perception of analogy as a learning tool

While analogy is the focus in this dissertation, this research is grounded in recognition of the CS1 course as a "whole system". Each learner is engaged with this system, and its design affects their engagement and learning. The novice learner's ability to learn in the classroom is not *solely* based in cognitive processes or pedagogical tools. These considerations for prior experience, perceptions, environments, and other factors are sprinkled throughout this work as a result of grounding in user experience design.

1.3 Organization of This Dissertation

Chapter 2 explores how **mental models** are utilized, and how they develop through learning. Factors that limit mental models are considered, as well as bridging the novice-to-expert divide.

Chapter 3 discusses the **value of analogy as a teaching tool** in STEM disciplines broadly, and in CS specifically. We will work to critically understand the "bad reputation" analogy has garnered in CS Education. Finally, we will propose factors that should aid in reducing noted concerns.

Chapter 4 **introduces a framework design tool** I have created to help instructors create well-formed analogies.

Chapter 5 will consider **engagement, interests, and "concrete analogical representations"**. This explores how relevance and engagement for learners affects the learning process. This chapter also considers the interests learners indicate having, and presents concrete analogical representations intended to promote engagement.

Chapter 6 introduces **internet memes as a relational structure** to engage learners with analogical reasoning. I designed a study around the understanding and interpretation of memes, and present findings. These are consolidated into key takeaways suggesting the utility of memes as a viable communication and reasoning tool for learners.

Chapter 7 closes with **case studies on CS1 experiences**. This also chapter highlights additional considerations in the "whole system" of the CS1 classroom that affect learners. To round out our case studies, we will end by looking beyond CS1 at how higher-level students engage with analogy in their courses. These observations by higher-level students support this research on analogy, and encourage future work in analogy across levels of computer science education.

Chapter 8 **summarizes the findings and considers future work**.

The Appendices contain **data, procedures, and definitions** that will be referenced throughout this research.

1.4 The Value of Prior Experiences

The representations of the world, linguistics, problem solving, and creativity that novices bring are not hindrances. They are reminders of the processes and considerations that experts often take for granted. Our abstractions and quick reasoning are built through connections of prior and new knowledge, with analogical reasoning being a key component. Assisting novices requires consideration of whatever abstractions their prior experiences have afforded them. Scaffolding extensible mappings to these prior experiences can aid novices in developing an expert's abstracted schema of programming. Greater growth and knowledge transfer in our discipline may require a critical lens toward that which we consider to be "the way it is". As Grace Hopper [75] once said:

> "The hardest thing in the world is to change the minds of people who keep saying, But we've always done it that way. These are days of fast changes and if we don't change with them, we can get hurt or lost".

To that end, I present this dissertation as my investigation into how learners connect knowledge about programming and engage with learning it. Through this, my hope is that we can enhance the experience of the novice programmer, and better promote their learning and growth.

Chapter 2

Visions: How Do We Understand Our World?

This chapter answers the research question: *What role, if any, does the use of analogy play in the development of effective mental models and learning?* The primary focus is on modeling knowledge, and how knowledge models inform learning and evolve through learning. While cognitive and learning sciences have numerous theories on the process of learning, these certainly do not mean education is a solved problem. There is still much to uncover, and aspects of the learner, the environment, and the material can (and often do) play a role. The exploration here of several theories and concepts will inform our discussion of learning in the computer science classroom.

2.1 What is a Mental Model?

A mental model is, put simply, a specific person's understanding of phenomena within the world. In the first chapter of the book Mental Models[54], Donald Norman[125] describes these models in greater detail. He notes two imperative actions mental models allow us to do:

- **Prediction** of expected phenomena and outcomes as we interact with the world.
- **Explanation** of our predictions — the ability to justify why we believe them.

In describing a mental model, Norman also positions it as part of a larger system of models.

2.1.1 The System of Models

A target system is the process or system of processes being learned. When teaching students to code on the computer, our target is that they learn and become proficient in this process. This is the existing system we aim to transfer knowledge of. [1]

[1]Norman specifically notes a **physical** system. While learning to program in a language is technically a means to interact with the target physical system (the computer), the language is also a fully designed system to facilitate that goal. Further, the process of typing code into the computer is a physical system, containing within it the requirement of understanding *how to type that code* for the operational system to succeed — forcing the non-physical component into a physical system. One can also argue for the most likely rationale: physical is intended as a synonym to "tangible" or concrete. Programming languages certainly are perceptible. This is the most likely meaning, given Norman's discussion of computers and computer software within the same piece.

A conceptual model is a correct way of representing some information about the target system. That is, the conceptual model must represent some subset of processes and entities correctly, even if it does not encapsulate all the nuances of the target system. A conceptual model should not promote incorrect predictions regarding the subset of information it represents. When we present a diagram of the hardware inside a computer to teach how these parts interact, the diagram is a conceptual model. It is a correct representation of the computer for some specific concept, presented to facilitate explanation and exploration of the concept.

A mental model then, is one individual's interpretation of the target system. This is cultivated through interactions with the system, and also informed by our existing models and conceptions. The mental model may not be a correct model of the target system, but its possessor must be able to reason through their use of it, even if it is fragmented. It must "make sense" through some degree of perceptional processing — even if it cannot make sense when additional factors come into play. When one believes that arrays start at index one in Java, not zero, this is part of a mental model of how arrays function. This model is not correct, but to the learner, they can reason about arrays and the use of them with it. The learner likely has pre-existing notions that suggest to start counting items at one, not zero. To the learner, this model makes sense, even though it is inaccurate.

Observer conceptualization (for easier terminology, I suggest observer model) is developed by an outside person (such as a teacher) to model their perception of another person's mental model. If a student is having difficulty understanding how to call instance methods, an instructor may investigate this issue to determine where the breakdown exists. They may review student artifacts such as homework responses, talk with the student, or simply observe. With this data the instructor is able to develop a sense of how the student may be thinking about the problem. For example, they may identify a student attempting to have a Dog object variable named "jake" use the method "bark" by calling Dog.bark(), rather than jake.bark(). This evidence allows the instructor to form an observer's model. In this case, the student's mental model may include the notion that a method is invoked using the name of the class rather than the name of the object. The evidence that the instructor has informs this conceptualization. The observer model is a representation of how they *believe* another person's mental model is formulated. This may also be incorrect. It is one person's attempt to understand another person's mental model and represent it in some way.

2.1.2 What Affects Mental Models?

There are a number of pragmatic limitations to mental models. Norman [125] notes the following constraints:

- **Technical Background.** One's level of prior experience impacts their mental model. Expert and novice models differ due to the limitations of knowledge exposure and assimilation. Experts have enough background to allow developing more robust models.
- **Previous Similar Experiences.** Existing background in different but similar areas can impact model development. Such ideas — present before any reasoning or learning activity — are described by Clement et al [17] as **preconceptions**. A novice to the IntelliJ IDE may still have *some* mental model of how to interact with the system from experiences in another IDE, such as Eclipse. A novice who has not used any IDE previously may develop a preliminary model of navigating an IDE based on experience with other software applications. Model development is influenced by our belief in similarity between previously encountered systems and new systems.

- **Human Information Processing Structure.** Human brains have limitations in how they process information. These limitations can impact the design and development of mental models. It may be too resource intensive for a mental model to encapsulate the target system fully, or to reason about all necessary parts in a problem solving task.

2.1.3 Properties of a Mental Model

Norman [125] provides details on properties that can be observed in most mental models:

- **Incompleteness.** The model is likely to lack attributes or relationships that exist in the full system. Learning "rounds out" a mental model's completeness when information transfers correctly. Incompleteness can exist at varying levels, from rudimentary details to sophisticated underpinnings of the system, and can exist in experts. Incompleteness may not stop a mental model from being "functionally correct" for a task. An expert may be able to write and debug code proficiently, but lack knowledge of nuanced features in the language. A Java expert likely knows that comparison of Strings should not be done with == (the equality relational operator), due to comparison of memory references rather than the textual value, as Strings are objects. The expert may not know that Java's storage of String literal representations via the interning process causes the == case to function similar to .equals in several instances. The unique handling of String in Java is often understood by experts, but only to a degree.
- **Limited Runnability.** We expect that our model of a system should be able to be run — that is, given inputs, we should be able to simulate and make predictions using it. Mental models often contain limitations and gaps that hinder the ability to run effectively. Novice programmers may struggle to predict output or pinpoint the cause of a runtime error. An expert may have uncertainty as to the result of an operation in a specific context. Experts may "run" models from their fields daily, but still experience difficulties and limitations when they encounter novel problem spaces. This difficulty crosses levels of expertise.
- **Lack of Use Leads to Forgotten Details.** Some mental models are utilized and strengthened, while others lie dormant. Human cognition contains both remembering and forgetting. Dormancy of mental models causes details to be forgotten. These details need not be lost forever, but show the fickle balance of our mental processes. We cannot remember everything at once, but what we don't remember or leverage often is easily forgotten. Experts may say they are "rusty" with a particular language as they ramp back into syntax they have not used in several years. Learners may forget details of specific syntax or algorithms during a semester away, the summer, or even during short breaks.
- **Similar Element Conflation.** Concepts and actions that appear similar can easily become conflated. Many programming instructors are familiar with the nefarious "if loop" of novices. The "if loop" phenomena is evidence of this mental model property. Seeing both an if-statement and a loop, the novice notes several similarities. They see that both contain a Boolean condition regulating entry, some form of action bounding (curly brackets in Java, indentation in Python, and so on), and bounded actions to execute only when the condition allows. For novices, the presence of these component similarities can cause conflation. Expert conflation may be "mucking" of syntax across languages. Is the correct term for a followup condition to an if statement: elseif, else if, elsif, or elif in the given language? Or does it not exist, instead being an else that *contains* an if within it? A multi-language programmer knows these all represent the concept they desire, but may conflate which is appropriate in a specific context.

- **Superstitions Stick.** Mental models commonly contain additional unneeded steps, which the model owner knows are unneeded. These additional steps may be easier to carry out than the additional effort running or updating the model may require. A programmer at any level may re-run a program that has no randomized values or inputs. Despite making no changes, they may do this just to "be sure it still works". This is clearly superstition, but it is easy enough to run the program again and double check its output. Despite knowledge that a program with no inputs should provide identical runs, the superstitious action persists.
- **Brain Cost Effective, Brute Force Forward.** Norman describes mental models as parsimonious, where extra physical work is preferred to strategy (requiring mental effort) that would remove the need for that work. The more taxing the task is for our brain, the more likely we are to develop a solution with extra unnecessary action versus doing more planning. Programmers across skill levels often "dive in" and begin coding, even when the problem would be better considered by breaking it into smaller methods. This is an example of parsimony in mental models — it is easier for the programmer to simply jump in and do the physical work of typing the code. Engaging the mental effort of planning the approach is taxing, despite planning being likely to save steps.

2.2 Ecosystems of Mental Models

Humans have many mental models, and the system of models we possess is crucial to our capabilities of reasoning [164]. One might believe each person possesses only a single mental model encapsulating their understanding of the world, but this is untrue. Multitudinous mental models bring new potential and considerations to our processing capabilities.

2.2.1 Microsystems and Macrosystems

Models can be both tiny and grand in scale. Computer scientists likely to have a "macro" model of how general computer programming works. Within this, there are likely several mid-sized models, which may include "the compilation process", "instruction execution sequencing", "data storage within physical hardware", and "syntax of X language". These mid-sized models can be isolated from the macro "whole" if needed, decoupled and observed independent of the "macrosystem" they are part of. "How do I create a variable in Java?" requires only knowledge of Java syntax. This model can also be applied to a broader system to understand how the instruction is compiled by the machine as well.

Modelling systems within systems continues toward greater specificity. A model of "syntax of X language" likely contains models of language concepts, including loops, arrays, functions, variables, and so forth. These models contain further concepts, such as an array model likely containing a notion of what an "index" means and how it is used.

Models are not only strict class inclusions either — concepts may map across to other models and the representations within them. Humorous jokes might map the programming language Java and the colloquialism for coffee, java. While the Java programming language may have a coffee cup symbol, the corporate logo is not the only thing making the joke stick. The rich ecosystem of models in mind is traversed, correlating such distinct domains as caffeinated beverages to syntactic cipher solely on homography. Homographs as a concept is a modelled system itself, based in our written and verbal languages. Auditory cues, visual systems, touch, smell — recollections and incorporation of these into our models allows connecting of seemingly disparate models. The smell of sulfur can be a component of models for both cleaning out a fridge and evacuating a home, connected by the phrase “a gas leak smells like rotten eggs”.

Mappings need not be between completely disparate models, but may cross within a domain as well. Iteration as a general concept may be a process model that is not specific to any language model. Instead, it connects across them, with connection to specific languages changing the model interpretation to fit that syntactic lens.

Glynn [63] suggests that expert knowledge is comprised of interrelated networks. This describes our mental model ecosystem well: networks of macro and micro models comprising sums of domain knowledge, connecting across domains, and in turn, representing our knowledge as a whole.

2.2.2 Multiple Approaches and Perspectives

We do not always have only a single model for any given idea or concept. Often, we may carry multiple models of an idea. Entirely disjointed ideas contained within the same target concept [67] can even be maintained. Novice programmers may have models of memory and reference types, but not yet recognize the connection between them. They may also have distinct models of debugging while coding. Talking to a peer, hand tracing, drawing a diagram, and talking to a rubber duck may all fall into the debugging categorization, but are distinct models of engaging in that process.

This encapsulates an imperative realization: mental models can and do merge [164]. While debugging, one may begin by hand tracing, then engage with a peer during the process, only for that peer to switch to drawing a diagram. This shows multi-model engagement, and the ability of each debugging model to be cross-utilized to solve a task. Utilization of more than one model within a relevant task can create merged models [164]. Professionals working in information technology are likely proficient in a multitude of software environments, possessing models of each (which may already have connections between them). If a new task workflow requires the use of more than one environment for completion, the professional will likely develop a specialized hybrid model of the processes required. This hybrid model is a merging of their existing workflow models.

Merging of models can take several forms, from “spawning” new models to class-inclusion of other models. In the prior example, models merged by pooling and sharing relevant resources, creating an offspring-esque new model comprised of merged components. These specialized models are a product of merging. However, merging can also encapsulate class inclusions to macro-models or of micro-models. The expert programmer with their model of loops that connects across languages likely began with a model of iteration formed in one language. As they learned additional languages, they modeled iteration in them as well. Perhaps terminology similarities, or the environment of programming caused mapping across models. Regardless, such a mapping comes to connect these two systems in mind. Eventually, they may merge to create a more general model: iteration as a “language-independent” concept. This model of course must still maintain some connections or modeling of language specifics for it be of use in practice.

In merging micro-models, one might consider again the novice programmer learning about reference types and system memory. The novice may have an initial model of both, but be uncertain relationships exist between them. Learning more about reference types (i.e.: in Java their value is the memory reference) begins recognition that these models have information that likely should connect. The novice’s model of reference types may begin to merge to include aspects of their model of system memory.

It is not uncommon and in fact quite viable to have multiple mental models for what might be considered a single process. Recall Norman's observations of limited runnability and incompleteness. This can be due to "missing portions" of a model splintering it into multiple models. Similarly, observations may also be interpreted as distinct phenomena. Linder [92] exemplifies this in the consideration of light, noting it is beneficial in some instances to model it as particles (such as to understand movement and reflection of light), and in others to model it as waves (to understand its curving around objects or the spectrum of visibility). Both of these models contain correct conceptions about the phenomena of light. However, these models cannot realistically be mixed in the same way as our prior examples. Individually the models are incomplete ideas of light, but also do not meld together to form a conception of "light as a wave-particle". These models are also distinct from each other, interpreting correctly differing phenomena for light. In attempting to solve a problem related to light-phenomena, model selection can occur. Based on cues from the problem space, one discerns if the mental model for wave or particle behavior feels appropriate in determining a solution. Models must be able to be ruled out using some parametric or heuristic measures, as Norman [125] states. We might imagine the wave and particle models exist as micro-models within a larger model of "light". Problem space details may provide indications convincing one to begin approaching the problem with a specific model. If the problem describes the color spectrum of light, "spectrum" may more readily suggest "wave". This provides a measure of confidence in choosing the "wave" model to attempt to solve the problem.

Algorithm design allows us to observe the phenomena of mental model selection in programmers. One may begin a problem with an iterative approach due to this algorithmic style being more "top of mind" for them, thus less mental work to consider. At some point they may realize their iterative approach is a dead end, or now too mentally taxing, and switch to a recursive model. Ruling out a model is not a "one and done" process. Through attempts to run and utilize existing models, the learner may abandon prior modelling and start anew. They may alternatively attempt to "tack on" a new approach after a previous choice dead-ends. In their observations from mathematics, Williams et al [164] noted the emergence of a dialectic appearing to connect multiple-model switching. This dialectic provided rationale for the subject in "changing course" while solving a single problem. 'Track switching" is likely a familiar phenomena for computer science instructors who have engaged in dialogue with learners as well. This switching is also evidence of Norman's [125] observation of ruling out models. When one model becomes non-viable or the problem space unfolds further, the learner may assess that a different model is more viable.

Ruling out models is not only at the beginning or end of problem solving — it is part of a continuum. Choosing to switch models may be triggered by salience of cues in the problem solving space and associating those cues with model connections. In the light example, "spectrum" may strongly correlate with the "wave" model of light for a learner. While solving the problem, the learner may create connections to the idea of movement, which is more correlated to "particles", prompting a switch. Switching and ruling out processes are certainly evidenced and can be observed. The phenomenological nature of model connection and probes for specific individuals makes it difficult to draw any conclusions with certainty on when and how switching occurs.

The incompleteness and even incorrectness in mental models can be a strength to human reasoning, as suggested by Williams et al [164]. Our capability to switch models mid-solving in order to facilitate task completion allows us to uniquely approach problems by adapting to present stimuli in the problem space. Williams et al identified dialectics allowing inaccurate mental models to merge and promote correct assumptions. This is critical for education: recognizing that disjoint models — or models with misconceptions — may not always hinder task completion in learners. Learners are able to reason, adapt, and adjust.

2.3 Evolution of Mental Models

We develop, merge, connect, adapt, and forget models in ways that allow us to formulate knowledge. The rich ecosystem of mental models contained within our craniums allow us to consider and reason about the world around us. As Jonassen [80] beautifully states:

> "How one constructs knowledge is a function of the prior experiences, mental structures, and beliefs that one uses to interpret objects and events" [80]

2.3.1 How Do Mental Models Change?

Our mental models are continually challenged, modified, and affirmed by interaction with target systems [125]. By engaging with phenomena and systems, our understanding of their workings adapts and revises. New connections are added and prior connections are affirmed. Connections lacking value are accessed less, weakening it within the model until it may even be forgotten. What we do and engage with causes our models to morph and grow.

Improvement of models happens through criticism and testing, as suggested by de Kleer and Brown [29]. As they state:

> "Much of the discussion has proceeded as if a person has *a* model of a device, when in fact he usually continually improves his understanding or model of a system [...]" [29]

The imperative should not be creating a perfect model of a system or device from the outset. Instead, the above claim posits that the systemic model continually improves. This sort of improvement should not be unfamiliar to programmers. We often design an initial solution space that represents our best effort to solve the problem. Interaction with the target system (the computer, and the software we program with on it), induces criticism such as incorrect output or error messages to the validity of our mental model. Our understanding of the system regularly encounters these walls in the debugging process, forcing adaptation of our original solution space to the target system's reality. In this way, our understanding of the target system and its processes can change. Such change may simply be at the level of that problem's solution, but can connect to macro-models of programming as a whole.

Novices may incite critiques of their mental models as they reason through programming by leveraging prior experience. In writing an array in Java, the novice programmer may attempt to access the first element. Their mental model helps them develop a solution to interact with the target system. This requires the novice to write Java code on the computer. They use index 1 in their attempt to access the first element. When running their code, the novice should see that the element accessed was not the one they desired. This interaction with the target system serves as a catalyst of criticism: *Why isn't this displaying the first element?* The novice must continue interacting with the target system in order to find their answer: the element they seek is at index 0, not 1. This process causes the mental model to adapt: the first element is at index 0, not index 1.

The process of mental model adaptation does not happen instantaneously. The novice may have this new knowledge top-of-mind when working to correct the error, but we must recall the properties of mental models. If the novice does not continue programming arrays and accessing the first element, this connection may not solidify. Instead, it may become a "forgotten detail" until criticism from the target system reminds them again.

An adapted idea may also be remembered, but may not "win out" to more highly connected information. One might wonder how a novice programmer would have any conception of where arrays should start prior to this that would "win out" in connectivity. The idea of starting, in association with counting is likely to be very strongly grounded in a model dating back to formative years. This model is how to count with whole numbers — which, for those unaware, begins with the number one. The counting model has likely been trod many times across a variety of contexts, making it strongly connected. Recall the parsimony of the mental model, which saves mental effort. The ease of accessing our general "how to count" information when beginning to program is likely to come to mind quite fast. Until connections to programming-specific information become more strongly defined, it is likely faster than "how to count when working with array indexes in Java" to consider.

Formulation of a question to experts may cause the activation of more "context specific" models. Novice programmers may develop the above issue in their code, become stumped, and ask for help. In the process of asking they may realize the answer to their own question. Developing the question requires using additional programming terminology in order to explain the problem to someone else. These terms may more highly active the connections, resulting in recall and correction without any outside input.

Conceptual models, given their basis in the target system, can provide appropriate criticism as well. An applied conceptual model can allow learners to achieve understanding of the target system. With their visualization tool, Jeliot, Ma et al [98] forced cognitive conflict that encouraged programming mental model adaptation. They identify that learners will not reconstruct mental models while they are still "happy" with them — as in, so long as the models remain unchallenged. This aligns with what Freire [46] suggests of critical pedagogy:

> Even if the people's thinking is superstitious or naive, it is only as they rethink their assumptions in action that they can change. Producing and acting upon their own ideas — not consuming those of others — must constitute that process. [46]

For our models to grow, we must engage with situations that pose criticisms toward them. While Freire's work does not surround mental models, it encapsulates well the idea of model adaptation. Necessary critique cannot exist if we are not actively engaging with phenomena and connecting conceptions in our models for ourselves. To be a passive consumer of information, or a "meek receptacle" [46], does not inspire true understanding. It may appear Freire is dismissing the idea that we can learn from others, lest we become mere consumers. This is false: he is promoting discussion as a tool to understand:

> problem-posing education regards dialogue as indispensable to the act of cognition which unveils reality. [46]

The Learning Dialectic of Mental Models

Interactions with the target system, tools that further understanding of the target system, and dialogue are all important stimuli for evoking necessary criticism. Asking peers and instructors for assistance may allow the crafting of specific conceptual models. These can allow targeted critique toward aspects the novice struggles to understand, allowing for the same opportunity to adapt and grow.

Target systems are what we teach about, and conceptual models facilitate teaching about such phenomena. Learners possess mental models, which are built and adapt as they learn about new topics or investigate known ones. Instructors develop observer models from watching learners interact with material. These contain assumptions about known and inferred knowledge. Instructors can then use these models to drive further learning. They may work to dispel ideas that seem to be hindering model development, or provide critique allowing misconceived models to be adapted. To do this, the instructor will likely need to develop new conceptual models to target what they believe learners are thinking based on their observer model.

The process of an observer's model creation and addressing this with a conceptual model is a feedback learning loop. An idea is presented in the classroom. Learners develop some form of model regarding it. The models are put to practice. Misconceptions are drawn out. The instructor observes the learners and considers how these misconceptions may have arisen. Using this understanding, they develop a new idea to present which counters the perceived misconception. The learners develop some model of that idea. This may result in assimilation of it, rejection, or even confusion as to how it applies. This new information is further data informing revision of the observer model. With it, the process of developing a new way of approaching the conceptual model to target the difficulty begins again for the instructor.

This cyclical refinement is a dialectic, as learning becomes a discourse attempting to reconcile the reality of the target system and the learner's mental model. The target system as a process is often unyielding, while human knowledge is adaptable. One may consider this didactic as a result. The learner's mental model cannot "teach" any truth to the target model. The dialectical power is found in that we do not teach the target model, but a conceptual model of the target. These conceptual models can be conversed about. Defining "gap closing" Lave et al [89] notes it as dialectical movement between the "expected shape" of a solution and information at hand, in pursuit of a solution. Both the learner and the instructor work together to close the gap of target system knowledge.

Conceptual models can be rephrased and re-framed to help a learner arrive at the truth of the target model. Information the learner provides through the conversation informs shifts the instructor should make to the conceptual model. The dialectic is in the discovery and refinement of misconceptions. The learner informs the observer model through their dialogue, and this informs some modification in conceptual model to address the perceived issue. The cycle of feedback and refinement by learner and instructor is the dialectic. This allows the learner to grow toward understanding of the target through negotiation and discussion of the conceptual model.

Programming is also unique in that phenomena are often already "fed" through a model before they are observed. Programming in Java is a specific target system, but learners often interface with this system through an IDE. This IDE is in and of itself a target system they must learn. It also provides a lens through which they can come to understand the Java language. Norman [125] also described the **system image**: the way a system is designed, ideally based in a conceptual model. Software should be designed with conceptual models in mind to facilitate appropriate learning of the target system. An IDE's system image is designed to facilitate a programming workflow. By interacting with the target system of programming through this lens, the system image may facilitate the learning process via grounding in a conceptual model.

Error notifications from IDEs approach a dialectic of sorts with the system image that fits with Lave et al's [89] observation of gap closing as a process between human and environment. This dialectic does have limits. The conceptions the system image may suggest are limited by the conceptual models it was created with. Interaction with a human, however, allows for the dialectic cycle to refine and iterate, as the discussion continues to re-position and reshape the conceptual model. This is not to dismiss the IDE's system image. It is an amazing benefit that systems can engage in this process at all, and is a boon for software design. Work in exploring further optimization of this IDE dialectic is also advancing this space [154]. Limitations do still exist, however, within the system image that may not be present in the human dialogue loop.

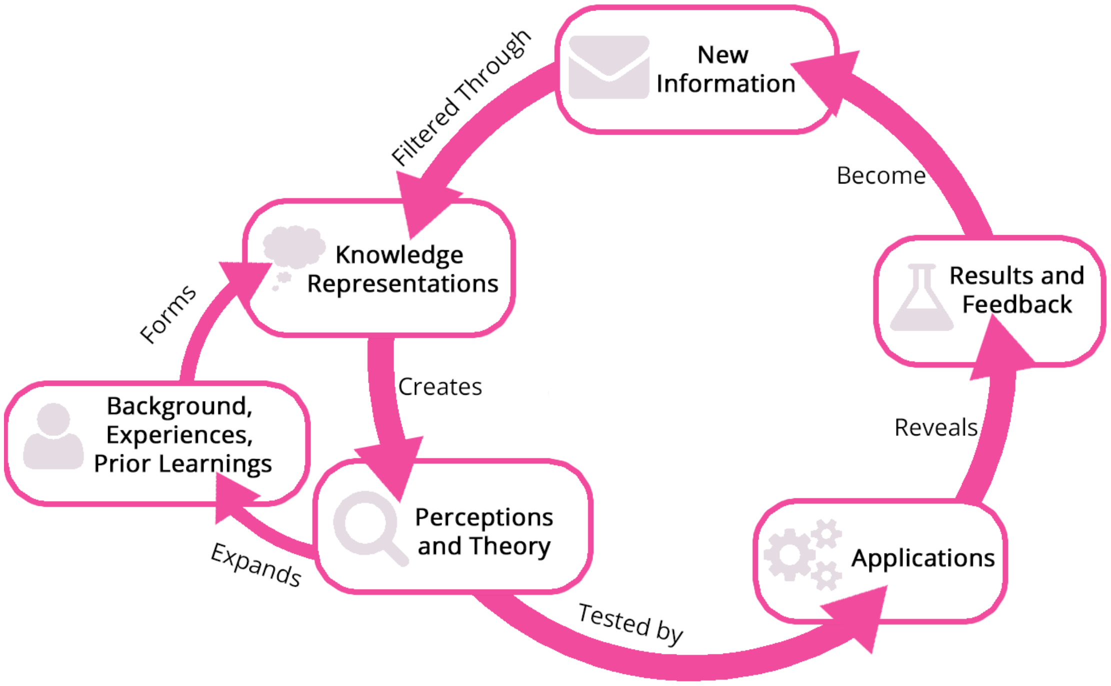


Figure 2.1: Diagram: Interpretation and assimilation of stimuli with mental models

2.4 Difficulties in Modeling

Prior notions, fragile knowledge, and context can all pose obstacles for the growth and adaptability of our mental models. Let us explore here how each of these factors might cause concern in the development of a mental model.

2.4.1 Existing Notions

The sum of our patterns and models form our conception of "how the world works", as suggested by Taber [149] — and this sum of knowledge impacts the design of our mental model ecosystem. These patterns have formed over time, and are what new knowledge or adaptations must become part of. Some assumptions we form — such as naive theories about physics from our existing models of the world — may be at odds with science [149].

Prior knowledge can impact framing of new information in existing mental models, or govern a newly created model's functionality [16, 52]. The understanding of "how to count" from grade school might influence perception of "how to count index values in Java". Clement[16] says:

> "Apparently one cannot consider the student's mind to be a "blank slate" in the area of force and motion." [16]

The sentiment that students are not "blank slates" holds true in nearly any discipline, despite Clement clearly describing physics. Even students with no prior programming knowledge are clearly influenced by existing mental models. This can extend not just to matters of counting, but linguistic interpretations of syntax meanings [146], or even how processes should execute. We will further explore some of these notions that may influence the novice programmer in Chapter 4.

Initial models will always influence new information, even if they contain errors, as Gentner and Gentner note [52]. The explanatory power of our models can be a doubled-edged sword. As they provide a capacity for explanation, we begin to believe that very explanation. Some beliefs may be easily dislodged through model critique, but others may be trickier to dispel. Beliefs may even be so deeply ingrained that the learner modifies information they have heard or learned in order to better align with their model, as McCloskey [103] observes.

Several learners may possess the same incorrect ideas, which suggests some "near-universal" conceptions instructors may need to counter. McCloskey, like Clement, examined physics students. As humans exist in the physical world, physics prompts many assumptions based in prior experience. McCloskey [103] noted not only the incorrect models that most students bring to the field, but also that many of these were highly consistent across students. Evidence suggested overarching naive model before beginning to learn physics. Programming would almost certainly differ as languages modify the experience. Any given language, however, might hold these overarching learner conceptions that oppose the system in practice. Technology's modern ubiquity may also prompt overarching naive conceptions about aspects of programming or the field of computer science.

Some beliefs may become so firmly held that it would take immense difficulty to revise such belief. Slotta and Chi [141] posited that this could occur within physics in relation to the *classification* of phenomena. Their work concerned the interpretation of phenomena as *substances* (heat being a property of an object) rather than as *processes* (heat as the transfer of energy). They saw that the former created a deeply held belief that was difficult to dislodge through traditional processes. So difficult in fact, that they suggested creating an entirely new mental model may be easier. However, Gupta et al [66] countered that while difficult, mental models of these exact processes can and often do still adapt. This aligns more with our existing understanding of mental model growth. They advocate that both learners and experts have dynamic models which are able to be adapted, challenging Slotta and Chi's conception.

Phenomenological primitives, or "p-prims" describe pieces of knowledge that are intuitive regarding the physical world. This concept is put forward by diSessa's knowledge in pieces theory [32, 33], which also suggests adaptability for physics students. These p-prims may require abstraction or modification for the novice to become the expert [32], but diSessa suggests that this existing knowledge is the foundation we must use to develop scientific understanding [33]. This fits well with the evolution of mental models as we have come to understand it.

We can also see that existing models affect new knowledge as errors in models are often based in plausible theories [16] — they do not come from thin air. The Java indexing error of starting at 1 is based in a perfectly plausible theory from prior knowledge — it simply is not correct for Java.

Describing misconceptions as “creative constructions”, Clement et al [17] describe them as likely adaptive and grounded in some real world success.

When given a target system, learners will implicitly project their own assumptions onto the system [29] — existing knowledge and ideas is not an escapable factor. Even if the target system is believed somehow “safe” from prior models, this process occurs. Signifiers and affordances [126] that the system presents are the reason for this phenomena. Given an environment or thing, assumptions are made about how it will work. Our mental models provide us the ability to predict, after all! Even in novel systems, our minds work to find some elements seeming to fit prior understanding. We use these to make predictions that we can then test and shape.

2.4.2 Fragile Knowledge Structures

“Fragile knowledge” was a term applied by Perkins and Martin [130] to novice programming knowledge which is incomplete, hard to retrieve, and often misused. Learning is a messy process: we work to adapt models, gain new information, connect that information, and often forget, misremember, or misapply other information. The three properties of fragile knowledge can be considered within our existing understanding of mental model properties:

- **Incomplete.** Incompleteness is not inherently a deficit to learning, as our mental models already possess this property. In this context it likely describes information not supplemented by any additional models. This puts further pressure on the runnability of the model. Programmers are often expected to be able to model and execute code, which causes this hindrance to be especially pronounced in programming.
- **Hard to Retrieve.** Difficulty in retrieving may relate to a mental model’s parsimony, as well as its lack of use. The “lack of use” for novice programmers is implicit in learning a new skill: they haven’t programmed prior to this point. With new information, the mental model is still becoming connected to our ecosystem of models. Retrieval is a taxing mental process. Retrieval without a high amount of prior access is even harder, as we will later see. The new programmer’s mental model tries to be economic in its use of cognitive resources, but this leaves less energy for retrieval of weakly connected models. This can also lead to forgotten details during the learning process — a byproduct of “lack of use”.
- **Often Misused.** Misuse aligns to the conflation of similar elements, and to superstitions. Concepts and actions, such as syntax and functions, appearing similar to the novice become conflated. This can easily lead to misuse. If the novice gets some block of syntax to produce the correct output, they may believe superstitiously it must always be applied in the same way, or take extra steps just to “be sure” of themselves.

The properties of fragile knowledge and the properties of our mental models align rather well. We can safely suggest that fragile knowledge does in fact represent a mental model. The connotation of fragile knowledge is that this is a model we do not want. This model, however, is part of the learning process. Instructors must recognize the fragile knowledge in order to challenge the model and inspire adaptation through targeted criticism. Fragile knowledge should not inspire fear out the gate: it can be a product of learning in progress.

There are three exemplifications of fragile knowledge which we must exercise additional caution toward mitigating. Perkins [129] describes these as:

- **Inert knowledge.** Concepts cannot be applied outside of the context in which they were learned.
- **Naive knowledge.** Reliance on explanations, formulas, or general observations that may not be applicable in flexible ways. We observed naive knowledge as an aspect of students' existing notions of physics in the section above.
- **Ritual knowledge.** Routines that work in the current context are favored over connection and application beyond it.

Inert, naive, and ritual knowledge may cause stagnation in mental model growth. Each suggests progression to a more connected and adapted mental model stalled. They can also indicate a learner is building a model of the classroom *context* as a problem — allocating resources not to learning the material, but to "whatever it takes to pass". We must be sure to recognize and apply criticism to learner models so that fragile knowledge can stay a byproduct of the learning process, not a result.

2.4.3 The Importance of Context

Context matters greatly to the development, growth, and recall of mental models. We build our knowledge in relation to specific contexts, and use salient cues to identify if models from existing contexts connect.

The frame of reference a learner has when solving a problem can modify assumptions created about that problem [103]. In education, this can be seen in the learner belief that the current learning topic must apply to the current problem. Given the classroom frame of reference, the learner contextualizes that the current topic should be relevant to solve the problem at hand. The learner may not yet know how they will use the topic, but the classroom frame shifts perception of the problem solving model. They may even fixate on the need to include the topic, neglecting other aspects of the problem space they otherwise could have solved. The selective attention caused by frame of reference in the classroom fits the old adage: "when your only tool is a hammer, every problem looks like a nail".

Framing may come from salient details provided by the problem space or examples that initially come to mind. As Gentner and Gentner [52] described, initial models can affect the way new information is perceived. The models that first spring to mind when working to understand a problem — examples, visualizations, known facts and processes — can present a frame through which we perceive the problem space. This framing is not inherently a limitation as mental models can be switched, merge, and adapt. It can certainly become an obstacle though.

Sociocultural factors are inherently tied to our mental models, containing our perceptions and expectations of the world and its phenomena. The notions and models a learner holds may be steeped in cultural factors [78]. Without appropriate dialectic in the learning process, learners may struggle to connect their existing sociocultural models and existence to pedagogical information. Freire [46] poetically shares:

> "To deny the importance of subjectivity in the process of transforming the world and history is naive and simplistic. It is to admit the impossible: a world without people." [46]

We must acknowledge in the process of education that learners are not tabula rasa. Their cultures and upbringing impact the way they view and the way they work to understand the world. The classroom environment and their lives outside the classroom create further social and environmental context.

Prior models are oft met with frustration by instructors, due to the resulting difficulty they can pose for one's pedagogical design. However, it is imperative to not immediately dismiss these prior models. In observing Polynesian navigators, Hutchins [78] notes their models are "illogical" if compared to traditional scientific models. Polynesian cultural methods despite this allow precise navigation. These models are also highly efficient, allowing for mental problem solving where Western navigators require technological aide. Hutchins observes that the knowledge we carry for navigation may be factually correct, but not all of it is inherently useful. In fact, utilizing much of it may be a mentally complex task. The Polynesian method of navigation may not be deemed "correct" by certain sociocultural standards, but results in precisely correct results for the given task easily. The fixation on replicating a model within learners can rob education of relevance in the learner's reality, instead making it an act of colonization [46]. As Hutchins describes in his concluding thoughts:

> Failure to take the utility of alien mental models seriously cheats us out of important insights. [78]

Understanding the sociocultural factors and models learners bring to the classroom allows us to observe a myriad of unique perspectives. The goal of the classroom setting is to prepare the learner regarding specific knowledge sets, but this does not mean that their models have no place. These models affect how our learners interact with the classroom and our curricula. Without attempting to understand the learner's sociocultural reality, we can appear disjoint from it, and make our material appear disjoint as well [46]. It is even more pressing that these alternative or prior models can change how an instructor views a student, which may impact how the students view themselves [16]. We must recognize the difficulty in the learning process, recall that errors are often based in plausible theory [103], and engage in dialectic to to understand our learner's models. This will allow us to teach in a way that more positively aids the growth and impact of theirs. We must recall that our *own* models are based in our sociocultural perspectives. As educators, we must remember our own mental models reflect perspectives we have integrated:

> how difficult it can be to get away from fundamental assumptions of ones culture. [78]

2.5 Where Do Notional Machines Fit In?

The notional machine is an idealized, correct abstraction of a specific computational process, intended to help learners understand that process [35, 34, 143]. Within computer science education, notional machines are frequently discussed. While the term is exclusive to computer science, the concept is clearly reflected in the conceptual model.

A box diagram to display array elements at indexes is a representation serving as a notional machine. A correct hand trace showing line by line execution and value modifications is also a notional machine. Such ideas would be defined as conceptual models within other disciplines. Each conveys correct information regarding the target system and are used (often by instructors) to facilitate learning about that system. Young [167] specifically characterizes them as conceptual models in his critique of "surrogate models".

Distinguishing notional machines as a discipline-exclusive concept can narrow the ability to draw from other disciplines, and muddy terminology. An instructor cares about the design of the notional machines they use in class, but learners do not have notional machines. They are learning, and the notional machine must be correct. The learner's machine is likely prone to errors as they continue to gather information — the learner's machine is a mental model.

"Learner notional machines" have been discussed in CS Education Research, despite this not aligning with the definition. To correct this dissonance, some have modified the notional machine definition as "the machine in mind that is executing". This would make the notional machine *the notion the student has* of the machine. This changes the property of notional machines being correct models, and aligns the notional machine to a mental instead of a conceptual model. Alternating desires in use of the term has caused context-dependent definitions that can muddy the intended meaning.

Utilizing a term specific to computer science education rather than a more broadly accepted term limits our ability to leverage advances made in other fields. Notional machine literature and exploration in computer science education has yielded valuable insights. Exploring conceptual models, however, is not inherently novel to computing.

Computer science education researchers use both mental model and notional machine as terms, but appear to rarely use conceptual model. This may be due to term conflation. Modelling methods such as UML (Unified Modeling Language) exist under an umbrella dubbed "Conceptual Models" in computer science. These methods allow the modelling of concepts within the program. This term is not wrong, but is also narrowly specified. A UML diagram provides a conceptual model with regard to a system's design. The UML diagram is not the program, but a mechanism used to inform understanding (and thus, mental models) of it. This narrowed specification is unlikely due to any dubious misappropriation but instead utility of terms. UML is a "modelling" language, as are the additional languages in this "Conceptual Model" umbrella. Model fits as a term. All fit to define a program's concepts, thus, they are conceptual. The specific usage in computer science likely grew entirely disjointed from fields such as cognitive sciences.

Correlation of notional machines as a target system [143] may have furthered term conflation. The theory behind this suggests students must learn the notional machine to execute code mentally. Thus, the notional machine is a target in the curricula. The notional machine is, however, still a conceptual model of the target system — an abstraction of the actual programming language or program execution on the machine. The use of the notional machine as a pedagogical tool to incite appropriate mental models of the system aligns it with conceptual models. The notional machine allows reasoning about code execution and creates a system to model this. That system is still a representation of some other system — such as the the "actual coding" for that language or machine. Thus, the notional machine cannot be the target system. It is a conceptual model.

All arguments point to computer science education's notional machines being the same as conceptual models from cognitive science. They are specifically conceptual models targeted to programming language semantics and program execution. The research and work on notional machines within computer science education is necessary and valuable — but it is important to understand how these terms interplay. The conceptual model exists beyond the domain of computer science education and cognitive science. In fact, it is present in education literature across STEM disciplines, and even beyond to industry. Pointing out the similarity between these two concepts can reveal the value of broader evidence across disciplines.

2.6 Human Faculties and Cognitive Considerations

Mental models have the pragmatic constraint of the limits of human cognition and the systems that govern it. This section explores considerations from cognitive science and neuroscience allowing us to better understand our mental processes.

2.6.1 A Brief Introduction to Theories of Intelligence

Theories of intelligence attempt to quantify and measure the cognitive ability of individuals. A rather thorough investigation of existing intelligence theories was compiled by Kaufman et al [133]. Showing the value to our present explorations, they state:

> An understanding of intelligence often provides insight into peoples capabilities, provides insight into why various psychological and educational interventions work for some people and not for others, and helps us grasp how affect develops differently based on individual differences in cognitive ability. [133]

Theories of how intelligence works allow us to grapple with mental models in different terms — specifically, terms attempting to quantify some utility in our mental model ecosystem via problem solving tasks.

Kaufman et al [133] put theories of knowledge into three major categories:

- **I. Theoretical Testing Foundation.** Theories in category I strongly correlate to measurement and testing IQ. Given the popularity of these tests, theories in this category are most likely to affect someone by classifying their intellectual status. Theories in this category include:
  - Cattell-Horn-Carroll (CHC) Theory [76]
  - Planning, Attention, Simultaneous, and Successive (PASS) Theory [26]
- **II. The Missing Pieces.** Theories in category II aim to "fill in the gaps" of category I. Notions of intellect which were not strongly represented in category I are advocated for here. Theories in this category include:
  - Multiple Intelligences Theory [47]
  - Successful Intelligence Theory [147]
  - Emotional Intelligence Theory [101]
- **III. Cutting-Edge Cognition.** Theories in category III are based in modern cognitive and neuroscience findings. They are difficult to apply practically at present, but have important scientific value. Theories in this category include:
  - The Multiple Cognitive Mechanisms Approach [84, 157]
  - Parieto-frontal Integration (P-FIT) Theory [82]
  - Minimal Cognitive Architecture [2]
  - Dual-Process Theories [40, 83]

Category III's modern scientific merit are valuable to understanding our learners, with notions from the Cattel-Horn-Carroll theory of category I providing relevant grounding and language. Each intelligence theory brings with it specific focus and assumptions. Our upcoming focus will align us both in the foundation of intelligence theory as well as the field's modern leanings.

2.6.2 Fluid and Crystallized Intellect

Two distinct entities in understanding knowledge exist as part of the Cattell-Horn-Carroll (CHC) Theory [76]: fluid intelligence and crystallized intelligence.

- **Crystallized intelligence** is the accumulation of knowledge one possesses. This can be facts, procedures, observations, and more. The key component is that this represents our accumulated "hoard" of knowledge about the world.
- **Fluid intelligence** is our ability to process new information and novel tasks, or the ability to abstractly reason about problem spaces. The key component of fluid intelligence is the novelty of the problem space. This makes it distinct from the accumulation of knowledge we possess.

Crystallized knowledge is formed from experiences, culture, and education. Meanwhile, fluid intelligence is the ability to abstractly reason and adapt [27].

Fluid intelligence and crystallized intelligence are separate "bodies" of intellectual capacity that do not cross in CHC Theory [76]. The theory also posits that while crystallized intelligence grows through life, fluid intelligence is largely fixed [76]. Modern neuroscience experiments into fluid intelligence have revealed that these capabilities can be trained [121, 131, 168]. This intellectual capacity is not so fixed as initially perceived.

CHC theory provides terminology for theories of reasoning and intellect, but *not* for learning. The astute reader may have realized the incongruity: **If fluid intellect and crystallized intellect never cross, how can we accumulate knowledge?** Fluid intellect allows us to reason about new problems, but crystallized intellect is our accumulated knowledge. Learning must occur at the intersection of fluid intellect and crystallized intellect. Where something stops being novel, it must start being incorporated into our knowledge. CHC Theory is for *measuring* intelligences, not explaining learning processes. The language of CHC Theory will be revisited in later sections, as we work to address this incongruity.

2.6.3 Working Memory and Cognitive Load

Working memory is not only a recognized cognitive system, but is also strongly correlated in neuroscience and cognitive research with fluid intelligence metrics [131, 168]. Working memory is a necessary component of reasoning about any problem space we encounter. Our working memory is limited, but allows information processing to take place. The exact nature of connection between fluid intellect and working memory, or if they are in fact synonymous, is still in exploration [168]. This may be due to working memory's system design still being largely debated. The following general conclusions are noted by Yuan et al [168] to exist about working memory among a majority of researchers:

- Working memory holds task-relevant information

- Working memory can source information from long term memory
- Working memory has several components contained within it. The specific components are debated. Proposed components include a visuospatial sketchpad for visual and spatial perception and phonological loop for auditory cues.

To succinctly define working memory (WM):

> [...] most agree that WM includes multiple subsystems working together to activate task-related information, maintain activation, and manipulate information during the performance of cognitive tasks (Miyake & Shah, 1999). [168]

A key component of working memory is its limitation. Working memory allows reasoning and solving in tasks, but is extremely limited regarding the amount of information one can reason about at a time. Common parlance maintains an original notion of working memory — Miller's observation of "seven plus or minus two chunks" [113]. This theory suggested working memory as a singular store of "chunks", rather than a system with multiple components. The utility of Miller's "chunks" still persists in mainstream discussion due to it being easy for the non-cognitive researcher to understand. At minimum, Miller's finding gives a basic mental model of cognitive load.

Cognitive load is the utilized capacity of working memory resources during a problem solving task. When one is "juggling" many pieces of information, cognitive load is high. The person likely struggles to remember all the information required of them for the task. They may make omissions or errors as a result.

Performance on science tasks drops when the information load given exceeded the capacity of working memory [168]. Understanding cognitive load and working memory is clearly important to education. Several authors have indicated evidence of cognitive load's effect on solving programming tasks, including Anderson and Jeffries [1], Muller et al [119], and Vainio and Sajaniemi [156]. Learners processing too much information will have decreased performance, as the load is too high for working memory to retain and reason about it all. It is recommended that educators (in fact, all experience designers) use methods for decreasing cognitive load [168].

Working memory and cognitive load allow us to better understand the parsimony of mental models. Cognitive actions are tasking, and limited resources are available to do such actions. As a result, models behave parsimoniously, reducing the load they place on working memory overall. Ideally, this frees up resources for more intensive aspects of the task at hand. However, it may (as we have seen) lead to less economical decisions.

2.7 Connection of Knowledge in Mind

All of the topics we have explored so far have required some form of **connection**: creating links or associations between ideas and things. This section explores the process of connecting knowledge, implications of knowledge connection, and proposes based on the literature a theory of describing the intuition of expert programmers.

2.7.1 Pattern Recognition

Pattern recognition is a central cognitive ability of human reasoning, and requires connection of ideas. Rather than explain pattern recognition, why not exemplify it?

```
Sally once tried to program in Perl
She thought she would give it a whirl
The syntax was rough
But ol' Sally was ________
So new knowledge began to ___________.
```

Were you able to fill in the blanks? Even if you were not able to, you likely engaged in pattern recognition through any attempt to try.

First, likely came some recognition of the syllabic structure as a pattern. You may not recall it by name, but you may be familiar with the limerick's AABBA rhyming structure. The rhyming of "Perl" and "whirl" couples with the line lengths. The first two lines are longer compared to the following two lines which is recognizable. Your mind implicitly connects: *"I have heard things like this before. This pattern is not new to me"*. The power of pattern recognition is further exemplified here when one realizes there is *not* full evidence of the limerick structure. With only AAB and line lengths being present. The presented writing still allows for a pattern to register in mind. Additional patterns limericks typically hold may also contribute, such as the introduction of a person and their traits in the first and second lines.

The limerick structure helps one identify that the fourth line must rhyme with "rough" and the fifth line must rhyme with "whirl" and "Perl". Almost immediately, we engage in a new form of pattern recognition: rhyme identification. The "-uhrl" sound and "-uhff" sound become a pattern we attempt to match within our vocabulary: *could it be gruff? Buff? Stuff? Fluff? Squirrel? Leg curl?*.

Puzzling through the potential rhymes engages one final pattern recognition: patterns of sentence and communication structure. "Stuff" rhymes, but makes no sense in following the pattern of communicating meaning. "Sally was stuff" adds no meaning, and has a grammatical structure that does not match what we know about language. We might expect that Sally "has" stuff, but "was" existing in the pattern template causes rejection. We may also see that "Sally was buff" could reasonably work as a sentence structure, but has no conceptual meaning. Muscle size is not corollary to programming capability.

```
Sally once tried to program in Perl
She thought she would give it a whirl
The syntax was rough
But ol' Sally was tough
So new knowledge began to unfurl.
```

Landing at "unfurl" is quite exemplary — it fits "Perl" and "whirl" only in the sense of rhyming. Where "Perl" and "whirl" are one syllable, "unfurl" is two. Further, all words carry the same "-uhrl" sound, but each spells it entirely differently! If you landed at unfurl over a one syllable match (such as "swirl", which meets all prior pattern requirements, and is a completely valid completion), congratulations on an additional layer of pattern recognition. Looking at the limerick as a whole one may recognize in trying to complete the final line that it "sounds" too short if it is not at least two syllables:

```
(Sa)(lly) (once) (tried) (to) (pro)(gram) (in) (Perl) - 9 syllables
(She) (thought) (she) (would) (give) (it) (a) (whirl) - 8 syllables
(The) (syn)(tax) (was) (rough) - 5 syllables
(But) (ol') (Sa)(lly) (was) (tough) - 6 syllables
(So) (new) (know)(ledge) (be)(gan) (to) ______. - 7+X syllables
```

Counting syllables, we notice even before the addition of the final world that the last line is still more syllables than the third and fourth. While the line "works" with a single syllable, it sounds abrupt when applied to the cadence of limericks. In trying to understand this, I explored several examples of traditional limericks [163] and counted the line syllables. I noted that another implicit pattern of limericks is that often, the first line contains equal or more syllables to the second, while the final line contains equal or more syllables to the first. This design supports the cadence of the limerick. Including a single syllable word violates this observable pattern while a two syllable word satisfies it. This would likely only be familiar to someone with a pattern of limerick cadence. [2]

Whether you filled in any of the words, got close, or completed none, this example showcases several ways we engage with the process of pattern recognition. Wherever you landed, you likely engaged at least one of the forms of pattern recognition shown above: recognizing the limerick structure, recognition and fulfillment of rhyming, or recognition of communication patterns and sentence structure to simply complete the blanks in a way that allowed inference of meaning. Each of these processes required pattern recognition.

In learning to program, a novice may recognize that two problems appear to have similar specifications, and thus, may have similar code. For example, take these two programming problems:

- Write a program that asks the user to guess a number from 1 to 10 until they guess the correct number.
- Write a program that prompts the user to enter a password. So long as the entered password is incorrect, allow the user to try again.

Both are "different problems" in terms of their domain and goals (fun games versus protection of information), but they contain patterns that can be identified, especially by the expert programmer. Both problems require: user input, repetition, and more specifically, repetition contingent on the input's similarity to some other information.

Pattern recognition is the process of matching stimuli information with existing information we already possess. We engage in this process when encountering new and novel information. This allows us to better process and connect it to our models of the world.

2.7.2 Abstraction

The ability for us to apply existing knowledge to new stimuli requires the act of abstraction. Where pattern recognition exists, abstraction is sure to be.

As Gentner and Hoyos [53] describe:

[2] Even with such familiarity, the pattern is difficult to explain. I wrote the limerick specifically for this section, knew all the "general" rules of limerick structure, and am a general fan of poetic verse. I knew a one syllable word didn't sound *quite* right, but still had to investigate to justify that inclination, only to stumble upon this reasoning. Hence I note the selection of unfurl as quite frankly, an exemplary case of pattern recognition by any reader.

> "We take the process of abstraction to be one of decreasing the specificity (and thereby increasing the scope) of a concept." [53]

In order to engage in abstraction, we remove some specificity from what we observe to allow it to fit with a broader scope of ideas. Abstraction requires focusing our attention on specific aspects of a complex whole [37], removing attention from other aspects. Let us take again the limerick example from pattern recognition. If one views the example as its isolated, specific self, it is impossible to match it to anything unless the match is literally identical. That would mean you have not only read those exact lines before, but memorized them and retained that knowledge. Barring that statistical improbability, you must engage in abstraction. Some specifics must be removed to allow the text to fit a broader scope. One might begin by removing the specificity of the words themselves in favor of the line structure and cadence. This may reveal the broader category of "limerick" can fit the text. To identify the pattern, we needed to abstract: specifics of the example are removed from processing to recognize a broader category of applicability.

Overhypothesis requires abstraction beyond the identified problem space scope [53]. This form of abstraction is one force driving the ability of our mental models to predict outcomes, even in novel situations. By generalizing some information about the problem space (first order), we may arrive at predictions based on applying those generalizations (second order).

Abstraction, like pattern recognition, occurs in a myriad of every day situations, and is often a near-autonomous process of our cognition. Imagine crossing the street at an intersection you have never used before. Despite never using that intersection, you likely do not hesitate to enact normal intersection-crossing routines. This is possible because the specificity of *that* intersection and its novelty is abstracted to allow it to fit the more general category of "crossing the street". This is due to recognition of familiar stimuli, such as streetlights and a road, which engage pattern recognition despite the novelty of the intersection. As we said, abstraction and pattern recognition go hand in hand.

Abstraction and pattern recognition are not only used with near examples, such as two programs or two intersections. These processes can also be engaged across domains, so long as the principle of relaxing specificity to increase scope is satisfied.

Abstraction can allow us to classify information within existing mental models. How though, did we achieve a broader conceptual category in mind that allowed us to classify information as a subset of it?

2.7.3 Analogical Reasoning

Analogical reasoning aids in our development of broader conceptual categories. This thinking involves reasoning about two ideas, examples, or scenarios and identifying the relationships between them. By recognizing these relationships, one can form broader abstractions that include both elements.

In learning to code one sees several snippets showcasing correct procedures. Here are some examples of Java variable declarations:

```
String name = "Jane Doe";
String greeting = "Hello World";
int someNumber = 5;
```

Analogically reasoning about these three lines allows one to form abstractions about general concepts. Comparing the first and second line, one may notice:

- Both lines begin with the word "String"
- Both lines have an equal sign, quotation marks, and a semicolon
- The word after String and the text in quotation marks appear related

From here one can also compare differences, attempting to develop an abstraction that fits both lines:

- The word after String is different on both lines
- The text in quotation marks is different on both lines
- The text in quotation marks also differs from the word after String

This process of comparing the two examples and their relationships can produce a more general conception of variables, which may include ideas such as:

- Variables begin with the word "String"
- They are identified by a unique word
- That unique word should relate to their contents
- The contents is in quotation marks
- The variable and its contents are separated by an equals sign

The third line presents new information that must be reasoned about:

- Does not start with the word String
- Contents are not in quotation marks
- The unique word is in fact two words, but no space is between them

The above information conflicts with some existing ideas in our abstraction. Here, we are able to see how a mental model may adapt. The novice programmer likely recognizes from the learning context that all three of these lines are related to variables. Their conceptions about variables need to adapt to include the information provided by the third line. Comparing the general conception and the third line may reveal:

- The text before the unique word appears to correlate to the variable contents
- Contents that are words have quotations around them, numbers do not
- The unique word may be more than one word, but does not contain spaces

This can cause adaptation of their general conception of variables to include the following ideas:

- Variables begin with a word that denotes their content type
- They are identified by a unique word or words, provided multiple words do not contain spaces
- That unique word should relate to their contents
- Textual content is contained with quotation marks, numeric content is not
- The variable and its contents are separated by an equals sign
- String denotes textual content, int denotes numeric content

The above scenario aligns with our understanding of how mental models evolve and exemplifies a way in which we develop more abstract general models of knowledge. These more general models constitute schema: a general structure of knowledge representing and encapsulating some information[168]. Analogical reasoning allows for the formation of schema [53].

Despite this power, analogical reasoning may not always occur in novel contexts [58]. If a pattern can be discerned, analogous reasoning processes can be undertaken [67]. The "if" appears to be the crux of analogical reasoning. In Gick and Holyoak's illuminating study [60], they had test participants read a story about an army attempting to invade a fortress, followed by having the participants attempt to solve a problem relating to tumors. If analogical reasoning occurs during their solving process (noted as "spontaneous analogical thinking"), members of the test group may recognize the story's information can be applied to the tumor problem. In the story, many roads led to the fortress, but a large number of troops could not travel a single road due to traps. To invade the fortress, small groups of troops entered from each road to not trigger the traps but allow all troops to arrive. In the tumor problem, a correct solution involves using laser treatment at a lower intensity from several points, so as not to damage the body due to a single ray of high intensity.

Viewing both of the last two sentences side by side likely shows the correlation of elements. A generalization of this problem might be summarized as "divide and conquer". However, readers of this dissertation are already in the context that we are discussing analogical reasoning. The instinct to compare both examples and identify patterns is prompted. In Gick and Holyoak's study, participants largely did not consider that the story they had read prior might contain information to help them solve the problem; thus, they did not reach correct solutions. When prompted with a hint to consider the story, the results changed. A majority of participants with the described story arrived at the correct solution [60]. To validate these results, two control groups were also tested. The first was a control-control group with a non-analogical story. The second was an experimental-control group with a story that promoted an improper solution through analogical reasoning. Comparison of the results showed that when the analogical reasoning was activated, participants of the test group were more likely to produce the correct solution [60].

While analogical reasoning is an incredibly powerful tool, a pattern's existence is not always discerned. The recognition of patterns is central to our human cognition, but uniqueness in our mental models coupled with our ability to abstract and recognize patterns can be limited. The ability to recognize and leverage these patterns is often considered an act of human creativity. Hofstadter [71] notes that creative solutions, especially in research, often ignore the boundaries of domain knowledge: recognizing patterns in areas beyond one's domain and applying those patterns to one's domain.

Analogical reasoning's core is the comparison of two entities or scenarios in order to understand their relationships and allow for abstraction. Given unknown scenarios or concepts, if a pattern is recognized and known information can be compared, we can even reason about yet-unknown states. This is overhypothesis [53]: using information to predict outside of our known scope.

Analogical reasoning underscores our ability to navigate novel situations. Consider a novice programming student entering the class computing lab for the first time. They sit at the desktop machine used in the lab. They have never used this machine, or even this brand of computer before. However, input stimuli about the machine are abstracted, and — especially given the context — they safely assume that this is, in fact, a computer. This allows reasoning and overhypothesis regarding how the machine works. Previous experiences with computers may have had a power button on the middle of the tower, causing the learner to instinctively look there first. They predict that in order to use this machine, they must find a button that turns it on. Further still, they use prior knowledge to predict the *location* of that button.

The brain is able to reason: "If X seems to be a case of Y, then what I know about X can also apply to Y". This punctuates the difficulty we experience as humans when using a microwave, shower, laundry machine, or oven that differs from ours. We may predict locations, names, or visual features of elements we interact with based on our existing mental mode. This can cause confusion when our prediction does not align with reality.

We can see how analogical reasoning is so core to our cognition: it allows reasoning about the novel stimuli we constantly encounter. Novel scenarios are commonplace in our human experience. When patterns are recognized, we enact these processes without much conscious effort. This process can be so readily enacted that belaboring how our brains process information about two different computers or microwaves might seem superfluous. We consider back to the observation that analogical reasoning does not always happen. What if we were unable to abstract that the "cancel" button on one microwave and the "stop" button on another shared common traits? We would be hindered from predicting what to do if we could not identify a relationship between the two. This may cause us to engage in more brute-force approaches to find the correct button. Analogical reasoning is only powerful if it takes place. Microwave to microwave or computer to computer may feel reductive, but programmers must consistently make analogical comparisons to complete their work. Novel and cross-domain contexts can make pattern recognition difficult [58, 60], but the ability to do so allows for effective problem solving processes to be enacted. This promotes connections in mind, more generalized schema, and potentially, new and creative perspectives to be achieved [60, 71].

Analogical Reasoning and Cognitive Function

Now that we have introduced analogical reasoning, we can connect this topic with fluid intelligence and working memory.

Neuroimaging has found high correlation between fluid intelligence and analogical reasoning [131]. Specifically, those with higher fluid intelligence were able to better manage mental resource allocation in order to analogically reason. Yuan et al's literature review [168] also noted that working memory and fluid intelligence are so closely correlated, it is debated within the neuroscience community if they are actually isomorphic or not. Using both of these as considerations, we recognize that if fluid intelligence correlates to analogical reasoning, and fluid intelligence and working memory may be isomorphic, all three factors must interplay.

Fluid intelligence requires splitting a complex whole into "simple, separately attended to parts" [37]. This decomposition aligns with our understanding of abstraction and pattern recognition, which are components of analogical reasoning. A specific whole is no longer considered, but specific aspects are attended to for recognition of patterns. Working memory is a limited resource, which may influence our decomposition.

Our conception of "simple parts" in fluid intelligence matters. For the expert programmer, designing a for loop might easily be considered a "simple part". It is one abstracted concept that is easily modelled and reasoned about alongside other parts of our complex whole. For the novice, however, a for loop is still quite novel. They juggle it not as a whole, but as its component parts: keyword "for", parentheses, what goes in the parentheses (three things, each separated by semicolons), what each of those parenthetical items are and their structure (initialization, condition, increment), curly brackets, and the inclusion of iterated code in the curly brackets. Taking Miller's heuristic of seven plus or minus two "chunks" of information, we can easily see how this makes the problem space a vastly different landscape for the novice than the expert. The novice is grappling with individual syntactic elements as novelties, trying to reason them into a for loop "whole" that fits the problem space. The amount of cognitive effort it takes to reason through just the design of a for loop might easily exhaust their working memory faculties! This highlights the importance of abstraction to schema-based knowledge. Abstracted knowledge allows for more entities and relations to be reasoned about as they become "simple parts".

In modelling working memory, relation of a node to the task's goal causes more activation, and prior retrieval results in lower activation thresholds [96]. This implies:

- we attend more to information that seems relevant to solving the problem
- it is easier to retrieve and use information we've used many times before

The adage of mastery through repetition holds for our working memory and cognition. As we "trod the same path" in similar problem solving contexts, it becomes easier to recall specific ideas and models. This effect can also hold *beyond* context — the more we retrieve something, the more readily we access it. Often, we find a correlation between our hobbies, interests, and memories to conversational topics. Information on these topics has been retrieved numerous times over our lives, making the pathways well-worn. Any stimuli we correlate to them through pattern recognition may easily activate retrieval. This ease of access ties well to our knowledge of parsimonious mental models — easy access reduces cognitive load in utilizing it.

The use of relevant information within the problem space promotes schema development. In reasoning between our existing information and the problem, we must recognize patterns and abstract. Thus, "practice makes perfect" again finds truth in cognition. A novice programmer seeing problems that require loops forces continued access to relevant loop-based information. Applying it requires reasoning about and adapting the model to obtain a correct solution. This reasoning and adaptation can result in a more schematic model of loop structures. The repeated habit of applying procedures allows past experiences (here, problem encounters) to form an integrated whole [89]. The more models of loops are leveraged, the easier they become to access and utilize. The ability to recognize their applicability is also heightened due to lowered activation.

Some Considerations Regarding Algorithms and Analogical Reasoning

Artificial intelligence (AI) and machine learning (ML) models within programming attempt to reasonably model intelligent decision making and thought processes. While this chapter has focused on human cognition, these systems are of interest, as understanding our human minds dovetails with programming such systems.

The act of mapping is an NP-hard problem without any constraints [51], and even when constraints are implemented [77]. [3] This mapping process is foundational for analogical reasoning in humans. Analogy engines such as Gentner and Forbus's [51] attempt solving a constrained mapping form modelling a subset of human cognitive capabilities. As human mapping must also include these processes, and since the analogy engines are NP-Hard, mapping in general must also be considered NP-Hard.

Mapping is also foundational in many AI and ML systems, and is especially necessary for ML recognition systems. Where humans can analogically reason across domains, AI and ML are largely limited to specifically designed domains. Mappings tend to exist within a given domain or set of domains. For example, This is due in no small part to optimization. Narrowing problem space and thus the domains, models, or representations an AI or ML works with can allow more efficient algorithmic solutions. Given the NP-hardness of the mapping space, effective and efficient solutions are preferable. Given system design (especially in commercial products), there is often little reason for the model to require the "full capacity" of human analogical reasoning.

Hofstadter's observation of creative solutions crossing domains [71] specifically calls out this limitation of artificial intelligence. He notes that AI problems are solved elegantly by limiting the problem space, but human-like creativity requires extension beyond imposed boundaries in pattern recognition. While not citing AI, Black [9] shares such sentiments in noting that analogical reasoning between domains is a core aspect of creativity in research. AI has not yet achieved what can be considered fluid intelligence [27]. AI systems that appear to possess some form of fluid intellect can be considered "task-transductive". The tasks for which the systems appear to exhibit fluid intelligence are actually isomorphic tasks to their existing design [27]. This means if the problem space differed the AI would not exhibit the same fluid intellect behaviors. If the system cannot generalize, the behavior cannot be considered fluid intellect.

These considerations show an intriguing convergence of AI and ML design with human analogical reasoning. Effective algorithmic solutions work to minimize the problem space, maximizing efficiency and correctness within a domain. This design may inhibit human-style analogical reasoning, but such inhibition may be necessary. As Wigderson [162] eloquently states:

> "The seemingly abstract, philosophical question: *Can creativity be automated?* in its concrete, mathematical form: *Does P = NP?*, emerges as a central challenge of science." [162]

The points made by Hofstadter, Davidson, and Walker are all rooted in human creativity: application of known ideas to novel problems to form creative solutions. As with many big questions in computer science, we arrive at the omnipresent P versus NP conundrum. Further explorations in this area of AI, ML, and analogical reasoning are beyond the scope of this work.

2.7.4 Analogy and Metaphor

Exploration of analogy and metaphor is imperative once analogical reasoning is fully defined: analogical reasoning is **any type of thinking that relies upon an analogy** [6].

[3] If any of the terminology in that statement is unfamiliar: the act of identifying and connecting relationships (mapping), even if limitations are added to narrow the problem space (constraints), still results in mapping being part of a set of problems widely believed — but — that no polynomial time algorithm exists for (NP-hard problem).

The comparison process for two similar and two very disjoint things undergoes the same cognitive reasoning faculties that employ analogical reasoning. One might believe the comparison of two obviously similar things, such as two microwaves or two loops, is wholly distinct from comparisons such as “a computer functions like a human brain”. This belief is not grounded in how our cognition works. Analogs need not be distinct domains, which is what often is drawn to mind by the term analogy. Comparing two loop structures in an attempt to understand how they function requires analogical reasoning. Each loop is an analog, and our comparison of the two allows us to recognize patterns, form abstractions, and predict. We can also reason using a generalized schema as one analog and a presented example as another. The concept of analogy applies to any comparison of analogs, and this comparison is the catalyst for analogical reasoning.

Recognition of analogy and the subsequent reasoning process involves four key subprocesses [49, 51]:

- **Retrieval:** A similar candidate must be identified in order to compare. This may be drawn from long-term memory or from external resources. A textbook may contain a strong candidate analog, for example. Finding and using that reference is external resource retrieval.
- **Mapping:** Correspondences between the candidate and problem space are made to reason about the nature of their relationship. This step “indicates ‘what goes with what’ [51]”. Mapping allows for the development of **candidate inferences**, enabling prediction and decision making about the problem space should the correspondences hold true. **Spontaneously run analogies**[67] are candidate inference predictions: we “run” the analogy as a model to draw conclusions on the problem space.
- **Abstraction:** Corresponding the two analogs leaves an abstracted, generalized structure of relational rules. This abstraction may be beneficial for further problem solving, and may become part of our mental models as a broader generalized model.
- **Re-Representation:** Partial mapping matches may require modifying the candidate or problem space in order to allow further mapping cohesion. This is part of the adaptation and evolution of mental models. Considering a mental model of a for loop alongside a for loop being coded may require revision of both the written for loop and the model of loop structure to better align the two concepts and solve the problem. Re-representing the solution in the problem space may cause our mental model to adapt in order to better align both analogs.

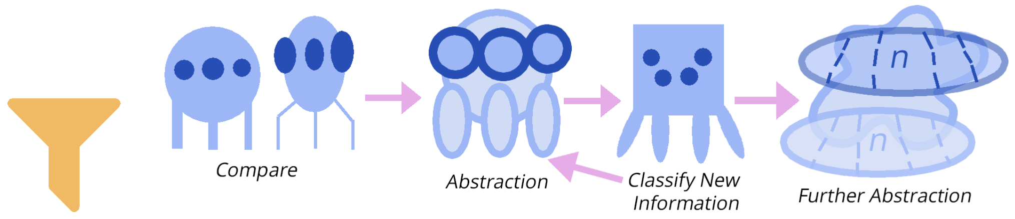


Figure 2.2: Diagram: Processes of analogy leading to mental model evolution

Analogy may not need to be linguistic [65]. We typically assume analogy takes the form “X is like Y”: “this loop is like that loop” or “technology is like magic”. While this formation is familiar, the linguistic component simply represents the comparative process. Reasoning analogically requires comparison of any two analogs, representing an analogy. Comparing the syntax of for loops is not inherently linguistic[4], yet we clearly analogically reason through this process. Visual or other sensory cues may also invoke analogical reasoning: "this department store's layout appears similar to my hometown's" allows candidate inferences regarding the location of items.

The way we might traditionally consider analogy is termed by Haglund as **post-festum analogy**, where one entity possesses information, and wishes to convey it by inducing analogical reasoning [67]. In stating "technology is like magic" or "knowledge is like stained glass", I hope to encourage analogical reasoning to incite some perspective. When we use analogy knowing the connections we wish to evoke, we engage in post-festum analogy. **Heuristic analogy** explores potential connections when topical knowledge is limited [67]. If in attempting to map "technology is like magic" one lacks topical knowledge of technology, magic, or any relation between the two, they may develop heuristic analogies in order to reason about the two concepts. These allow exploration on the "fit" of different connections to develop relational understanding. A heuristic analogy is experimental mapping, allowing hypothesis on connections so the best ones may be established. In heuristic analogies, we are uncertain of the relations and thus experiment in exploring them; with post-festum we know what relations are presented.

In using analogies while problem solving, one has the problem in mind, guiding mapping inferences [74]. Put simply: context matters! Heuristic analogy typically explores relations that fit solving the current problem. Someone clearly struggling with their computer might describe technology as magic. A programmer who recently completed a difficult but rewarding project and feels exhilarated may also describe technology as magic. In both cases, our interpretation changes due to context. The former is likely comparing technology to occult witchcraft, while the latter is describing gleeful enchantment with it. When presenting a post-festum analogy, our context inspired our recognition of intended inferences. This context distinction separates post-festum and heuristic analogies. In post-festum analogies, the context lies with the creator, while in heuristic, the context lies with the interpreter. This also means that what is a post-festum analogy to one person (the provider) may result in heuristic analogies for another (the recipient). When contexts diverge or are inappropriately interpreted, incorrect inferences may result.

Analogical reasoning that crosses **domains** (distinct areas of knowledge), considers a source (vehicle, base) domain and a target domain. The target domain is the goal of the analogical reasoning: we wish to understand more about the target through this process. The source domain is leveraged to facilitate the analogical reasoning. Using analogy in this way is projection [53], where known information helps us better understand the unknown. Consider this example: "hidden away, she was like a cocooned caterpillar". In the example, the source of a cocooned caterpillar is used to understand something about the target woman. We anticipate our knowledge of cocooned caterpillars should somehow apply to the woman. Knowledge of cocoons can pose two very salient conceptions: a barrier from the outside world, and a catalyst for change. We may reason that the target is undergoing some form of protection, a process of change, or perhaps both. Pointing again to context, prior knowledge may change which interpretation we lean toward. "Under watchful eyes, hidden away, she was like a cocooned caterpillar" lends toward protective, while "Immersing herself in her craft, hidden away, she was like a cocooned caterpillar" brings the metamorphic relation to prominence. Analogy never exists context-free [71] — even if no context is present, a person will develop their own. You likely formed context regarding the meaning of "hidden away" even before reading the situated examples!

Distinct source domain analogies can allow highlighting of different traits [52]. In Gentner and Gentner's work, they compared electricity to both flowing water and crowds of people. Their findings

[4]semantically yes, but not in the fundamental sense Guarini et al are indicating

noted that given these distinct analogies for electricity, participants reasoned differently about problems regarding electricity. Specifically, they employed reasoning based around what their analogy suggested [52], exemplifying candidate inference. Nersessian [120] identifies that there is no limitation on the number of source domains we might leverage while approaching new knowledge in an authentic manner. Similarly, Spiro et al [145] suggest that quirks of a specific source domain can be mitigated with multiple analogies. There is no necessity that analogical reasoning focus solely on a singular example. Similarly, there is no necessity to focus on a singular domain. We often have multiple mental models supporting a single concept, which was exemplified by Linder's description of light as both particle and wave [92]. In considering Gentner and Gentner's study, having both sources for electricity — the flowing water and crowds of people — allows varied perspectives and models of the concept. It can also promote generalized understanding by giving another source to form abstractions and re-represent understanding with.

Bridging analogies are developed in order to promote stronger confidence in a relationship or idea [18]. This can allow movement toward the target knowledge through incremental abstraction and re-representation. This aligns with multiple sources allowing resolution of discrepancies across sources. Bridging analogies differ slightly through the suggestion that target knowledge be adapted through incremental resolution of similar cases that align toward the target knowledge. This fits the concept Gentner and Hoyos describe as **progressive alignment** : abstractions with some sort of "concrete" general structure are easier to further abstract [53]. Such alignment is beneficial to learners, as it can allow them to break away from conservative generalizations: hesitancy to apply abstraction where it might be useful as the case is not recognized as generalizable [53]. Adding to its value, Clement notes that analogical bridging can occur among experts and lead to novel solutions [19] — this fits with Hofstadter and Black's thoughts on creative thinking [9, 71].

Progressive alignment of many examples can make distinct elements come to seem literally similar [95]. As we develop "concrete abstractions", we come to see elements fitting those abstractions as literally similar cases. Gentner et al [58] notes that encoding patterns change as our domain knowledge increases, and Qian and Lehman [132] affirms within computer science that expertise progresses from less to more abstract. Together, these results suggest that progressive alignment promotes continued abstraction until distinct cases can easily be considered "literally similar". These results also suggest that our encoding changes through this schematic development process. Holyoak and Richland [73] suggest that analogical reasoning can lead to analogical "bootstrapping", in which a schema allows for inductive reasoning and further abstraction. This promotes continued schema transfer [58] and progressive alignment [53]. Appropriate generalizations allow more cases to work with our schema, thus allowing for greater applicability of the schema, promoting abstraction and re-representation with it.

Comparisons such as a loop to another loop do not require source-target conceptions as they are within-domain. Still, they are not immune to context and its effects. A novice programmer who has only seen examples of booleans used in a control flow structure may infer from the context that boolean is only relevant "within the parentheses" of an if statement or loop. Reasoning about the examples creates context which frames their knowledge.[5]

[5]This may seem to the expert to be an absurd example. I would have thought so too, had I not experienced it firsthand on several occasions. I have observed students struggle to reason that boolean is a type they may need to use for a method/function's return (despite knowing that yes, it is a variable type) or even question what utility a boolean variable could possibly have. Examples in resources and assigned work are often heavily contextualized to show boolean used for control gate behavior (especially in CS1 to exemplify relational logic), that learners form their own context of where boolean "belongs".

So What Is Metaphor?

The title of Gentner et al's chapter in *The analogical mind: Perspectives from cognitive science* best sums up metaphor: "Metaphor Is Like Analogy" [56]. Gentner et al note that metaphor is cognitively processed much like analogy for novel metaphors, but not conventional ones [56]. These classifications will be elaborated on as we explore the Career of Metaphor in the next section.

> "The locus of metaphor is not in language but in the conceptualization of domains." [88]

Folk generalizations of metaphor versus analogy tend to focus on whether "like" or "as" was used. We are taught to distinguish analogy from metaphor if such words are present. This logic dictates that "I am like a mother bear" is an analogy, while "I am a mother bear" is a metaphor. The conception of embodiment, "becoming", is stronger in the metaphor. However, unless we legitimately believe that a bear has learned to write and shared this sentiment with us, we identify the relational structure as largely the same. This means we can easily interpret the analogical derivative ("I am like a mother bear") from the metaphoric form. As Lakoff suggests in the quote above, the language we use does not govern metaphor's "power". Metaphor's power lie in the way it creates mental imagery and consideration across domains. This power is availed to analogy as well. One difficulty of metaphor and describing it is its range of representation [52]. Take for example, the embedded metaphor in Maya Angelou's "Still I Rise" [3]:

> "You may shoot me with your words,
> You may cut me with your eyes,
> You may kill me with your hatefulness,
> But still, like air, I'll rise." [3]

Angelou's metaphor is a much different construct, not following the same "form and function" of the "like or as" generalization. We cannot so simply morph these words into the analogical format, and to do so would deny them their beauty and power. This form still allows Angelou's words to create a system of relations: we are able to map the acts of the antagonizing party (words, eyes, hatefulness) with acts of senseless violence (shoot, cut, kill). This powerfully illustrates the harm implicit in these actions. What's more, we are able to connect two disjoint domains across each line: one does not "shoot" another with words or "cut" another with eyes. The analogical relation is embedded in the line: "your words are like a gun"; "your eyes are like a knife" — and we are able to reason on this relation despite the absence of any firearm or blade predicates.

This shows the true power Lakoff suggests, exemplifies the range that Gentner et al note, and shows how deeply tied metaphor is to analogical reasoning. Angelou's stanza requires reasoning about the relationship of cutting and eyes not only on that single line, but as part of the overall piece. We create a map of relations allowing us to abstract generalities and gain understanding. This is the novel metaphor case for which it was noted that metaphor in spite of its representational range ultimately functions like analogy. Metaphoric reasoning is natural to our cognition: metaphoric reasoning often begins before literal reasoning has ended [62].

Gentner and Bowdle [50] posit that the structure "X is like Y" promotes a comparison of similarities, while "X is Y" is a statement of categorization. Categorization requires considering how can X be included as part of the set that defines Y — in what ways is X an instance of Y? Gentner and Bowdle also observe that the act of comparison promotes categorization over time [50]. This fits with abstraction and re-representation: comparison of two analogs promotes generalized abstract structures. The structure represents the intersection of both analogs, and can be used for categorization action over time. This allows recognition that some entity must also be able to exist within the same structure.

Metaphoric coherence is the way in which various metaphors interrelate to each other [100]. Single concept can often be structured by a number of distinct metaphors [88]. "It was an uphill battle to complete this project.", "I need to recharge from all that editing, I'm out of juice", and "Any more revisions might just push me to the limit" are all distinct metaphors for the concept of a difficult project. This aligns with no limitation on source domains for analogical reasoning. Each of these metaphors draws distinct domains, but all facilitate coherent understanding of related ideas surrounding the project. Each metaphor is abstracted and understood in isolation, but also as parts of a larger whole.

Career of Metaphor

To better understand the range of metaphoric representations Gentner and Bowdle [56] put forward the career of metaphor model. The model suggests that metaphoric ideas undergo four stages:

- **Novel.** The metaphor is new. We must analogically reason and compare to understand its implications.
- **Conventional.** Repeated application of the metaphoric representation, allows us to associate multiple meanings with the concept: our original meanings and the novel ones. For example, "cold" can relate to both temperature and disposition. Metaphor becomes a large source of polysemy in language as a result [50].
- **Frozen.** The metaphor has become highly conventionalized, able to be used without awareness metaphor has been leveraged. The original, literal conception can still be traced. However, the relationship to the literal idea is overshadowed. "Time is like a river" is an example. The concept of a literal body of water is present, water is overshadowed by the concept of "flow".
- **Dead.** The metaphoric meaning is now understood as literal meaning. The original literal meaning is no longer common in general parlance.

The floppy disk as iconography in interfaces is an example that can allow greater understanding of the career of metaphor. The concept of "saving to disk" was originally novel for many users. The image of the floppy disk was used to associate the object needed and the action desired as a result (skeuomorphic mapping by experience designers at the time). This use was novel metaphor: one had to map the association of the physical floppy disk and the floppy disk icon to recognize the "save" action was being signified.

Through repeated exposure to the floppy disk icon metaphor, the icon became more conventional. Inferring multiple representations was easier: floppy disks are both "a physical disk" and "an image to indicate saving to disk". The floppy disk thus became conventional metaphor.

Time went on, technology grew, and the floppy disk became a frozen metaphor. Floppy disks became less common, but the icon was still widely understood as the "save to disk" action. One could still recall the roots of the icon: the physical object and its relationship to the action its icon representation encourages.

The floppy disk finally became dead metaphor for later generations. Modern generations are often unaware of what a floppy disk *is*. They recognize the physical item as a representation of the "save icon". The floppy disk became associated only with the metaphor its representation conveyed.

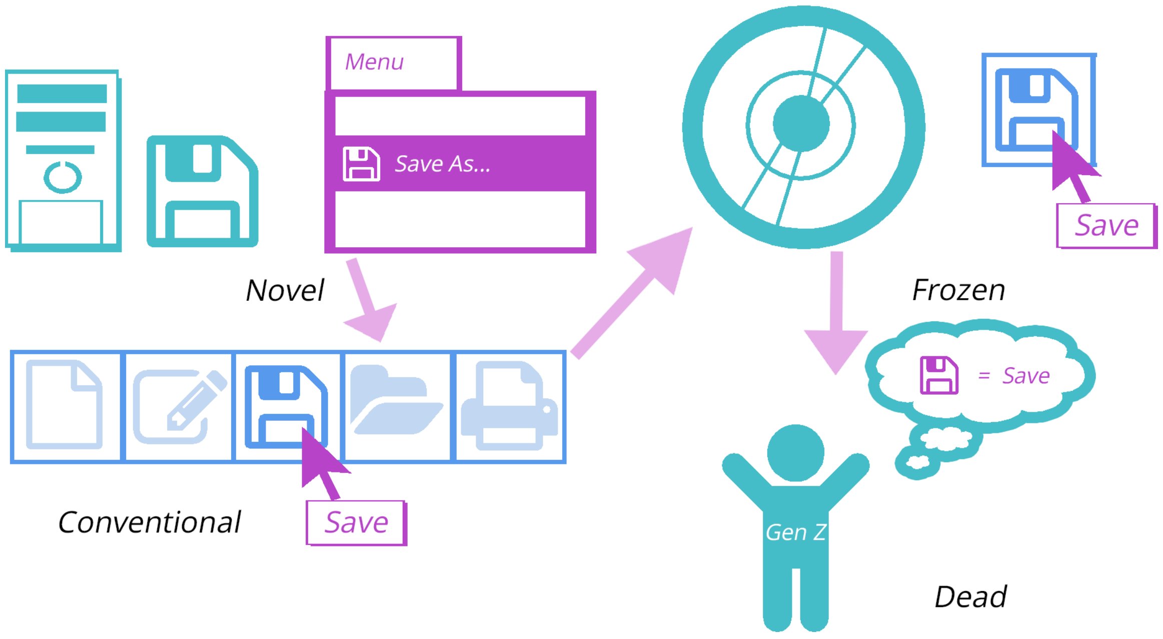


Figure 2.3: Diagram: Floppy disk as an icon's progression through career of metaphor

Gentner et al's indication that metaphor behaves like analogy when novel, but not when conventional [56], is now clearer. Novel metaphors are new to us, and we must reason about the domains to draw inference. Through repeated application, the metaphoric meaning becomes implicit. As a metaphoric idea becomes conventionalized, we understand it as a second meaning, so there is no need for analogical reasoning. We understand X *is* Y, or at least among Y's definitions. Even the simple word "line" [72] is highly conventionalized: we near-instantly deduce which of its many meanings apply based on context.

2.7.5 Structure Mapping and Systematicity

A **morphism** is an abstraction derived from a structure preserving map[65] — this notion of maintaining structure in the mapping process is necessary for well-formed analogical reasoning. Structure mapping is a renowned cognitive theory of analogical reasoning proposed by Dedre Gentner [49]. The theory posits that analogical reasoning processes prefer relationships, not surface aspects. To quote Gentner:

> "Relations have priority over object-attributes in analogical matching." [49]

Analogical reasoning requires analogical matching: the identification of how exactly items X and Y are analogous. We must map aspects of X and Y through a process called structural alignment [53].

This mapping of X and Y develops an abstracted common structure, which must hold true for both X and Y.

Consider a computer science example: “a priority queue is like a hospital emergency room”. We don’t intend for physical properties of the hospital to be mapped to our queue. Thinking of patient capacity and queue capacity isn’t all that helpful. When relationships are mapped instead, the reasoning is much clearer:

1. priority queue CONTAINS data; emergency room CONTAINS patients
2. priority queue CAN-GAIN elements; emergency room CAN-GAIN patients
3. priority queue ASSESSES priority flags; emergency room ASSESSES symptom severity
4. priority queue FIRST-ATTENDS-TO highest priority; emergency room FIRST-ATTENDS-TO severe symptoms
5. priority queue OTHERWISE-ACTS first in first out (FIFO) order;
   emergency room OTHERWISE-ACTS first in first out order
6. priority queue REMOVES attended to items; emergency room REMOVES treated patients

Mapping based on relationships allowed an abstract relational structure to form between the two concepts. This leads to the systematicity principle, which Gentner proposes as:

> “A predicate that belongs to a mappable system of mutually interconnecting relationships is more likely to be imported into the target than is an isolated predicate.” [49]

In our example, each of these relations are quite interdependent on each other. In order for 2 and 6 to be true, 1 must hold. For 4 to occur, 3 must also exist. In the case that 4 is not relevant, 5 interrelates to 3. Similarly, if 3 results in several identical levels of priority, 4 and 5 must be considered together.

> “by promoting deeply nested relational chains, the systematicity principle operates to promote predicates that participate in causal chains and in other constraint relations.” [49]

Systematicity enters the mapping process of analogical reasoning [55]. This is further backed by the observation that when reasoning about analogs, systemic relations are preferred [51]. However in long-term memory, surface matches dominate [51]. This may seem to conflict with prior observations. However, it is a well-reasoned cognitive strategy [51] as:

- Similar looking things likely behave similarly
- Representations in mind often have more salient surface properties we have retained

When we reason, we prefer structural relations; when we retrieve, we prefer surface similarities. This implies that surface similarity can certainly impact the choice of analogs we reason about. Before we begin mapping, surface cues may cause nodes to be more easily activated [96]. As we map, systematicity switches our focus to associating relationships in order to assess structure.

Even if surface features outnumber relations, we are still capable of structural reasoning. Forbus and Gentner [43] describe representations as top heavy and bottom heavy. A representation is **top heavy** when most ideas are relational, with less surface details. **Bottom heavy** requires that there always be a greater amount of surface information than relations. It is possible for structural commonality to be found in bottom-heavy representations despite a greater number of surface similarities. This means that relational schema development is still possible. When a generalized schema is developed as a representation and committed to memory, this can also serve as an effectively activated node for future retrieval [58].

While surface similarity primarily dominates long term memory retrieval, hope is not lost for relational retrieval. If analogical reasoning allows relational schema to form, they are both more likely to be retrieved *and* more likely to be further abstracted and encoded. Promoting retention of relation-based schema promotes recall despite our preference for surface similarity. Gentner et al further suggest this allows a path to inert knowledge. Inspiring analogical reasoning on prior knowledge and events can promote relational schema development while learning [58]. This may also help learners "look back" and identify additional examples, re-illuminating prior knowledge. Spotlighting relations through comparison mapping (and thus systematicity) changes how we reason about the past and encode future information [58]. Major impediments to the learning process are inert knowledge and passive learning [10], and so connecting inert knowledge meaningfully is promising.

Surface versus structural distinction relates to our goal attainment [60]. When comparing distinct processes or entities, what is structural versus surface may change. For example, being the color red is likely a surface similarity when comparing two laptops, but may be a structural similarity when comparing two mushrooms — as it would allow identification of both as poisonous/hallucinogenic. Context is again imperative: what we view as relevant in terms of structure directly relates to the goal we have in mind and the problem space we are engaged with. In addition to structural pressures promoting systematicity, there are also semantic and pragmatic pressures to analogical reasoning [74]. Semantic pressures encourage similar meaning across elements. Pragmatic pressures lean correspondences toward what is most important or viable for the reasoner. Cognitive load and working memory place limitations on our reasoning capabilities, which feeds into pragmatic pressures. Our context and existing knowledge may further limit, skew, or direct our reasoning toward what we see as imperative.

The act of structure mapping can yield three key outcomes beyond abstraction [53] that promote the evolution of our mental models and knowledge:

- **Candidate Inference.** Based on our mapping, we infer aspects about the problem space. This allows us to make predictions or **overhypotheses** about additional relations. "If this and this hold true for both, then perhaps this also holds true"
- **Alignable Difference.** Differences that behave similarly within both structures are more prominent.
- **Re-Representation.** One or both analogs may be modified in order to better improve the success of mapping, as previously observed.

Structure mapping and systematicity allow us to adapt our existing mental models effectively. The development of generalized relational schema allow greater applicability in new scenarios. This allows us to form meaningful abstractions that advance our general understanding. Repetition with this mapping process evokes true change and evolution in our mental models. We move from our immediate concrete problem spaces, toward deeper implementations, promoting further connection and broadening of our "macro-models" of associated knowledge [95].

> "Structure mapping is not just about identifying that something matches something, but specifically how. And as we figure out how, we may override some of the initial correspondences to fit the overarching understanding. " [95]

2.7.6 Towards A Theory of Intuition

"Intuition", and how the novice to expert mindset develops, is an area of intrigue for computer science education. Based in this chapter's research, I present some ideas that may help us move toward a theory of intuition.

Hermaneutics describes the whole being as a synthesis of its parts, but that parts infer context from the whole. Novice programmers grapple to break the whole into parts, but understanding the parts becomes so intensive that the whole is lost. The cognitive load required to design a for loop takes so many resources that the reason *why* the for loop was being written can be forgotten. Writing programs is novel, and taxes our fluid intelligence.

Fluid intelligence applies to reasoning about novel problem spaces. A problem space must become non-novel at some point, and part of our crystallized knowledge. The pieces we understand and crystallize at the beginning may be small "fragments". These "fragments" might be ideas like the keyword "for" relates to iteration. The general structure of the concept is lacking as the "fragments" are small and disjoint. Working memory fills when reasoning about these small "fragments". Cognitive load is heightened and focus in the problem space narrowed.

Continued application to similar problem spaces causes the same fragments and processes to be activated during reasoning. "Fragments" begin to connect together in more abstract generalized systems. What was once a distinct piece is now many pieces connected, a "cluster". This "cluster" is still one entity, just as a "fragment" was. The cluster, however, contains much more information within it. The "clustering" reduces our cognitive load: many interconnected elements now activate with the same amount of cognitive effort a single element used to take.

Continued activation of the same "clusters" and "fragments" in similar problem spaces carve "pathways" for fluid reasoning. The activation threshold to this information is lowered through repeated use. Just as a river carves a path, our fluid intelligence carves a path to the relevant nodes through repeated application.

Eventually, the path is so deeply carved and activation so lowered that the expert appears to have "gut feelings" or intuition. Repeated activation of the relevant structures in similar problem solving contexts yields "instinct". This activation and reasoning has formed "fragments" to "clusters", and "clusters" to entire "caverns" of abstract relational models. We reason easily about them, as they become single representations in mind. This accumulated activation and abstraction over time leads to incredible ease with complex processes.

The novice experiences programming as a novel task. As the task becomes more conventional, it becomes easier to map each part's correlation to the whole and to understand the whole's breakdown into parts. We come full circle to the implication of hermaneutics, and better understand how our mental models may develop into the phenomena described as intuition.

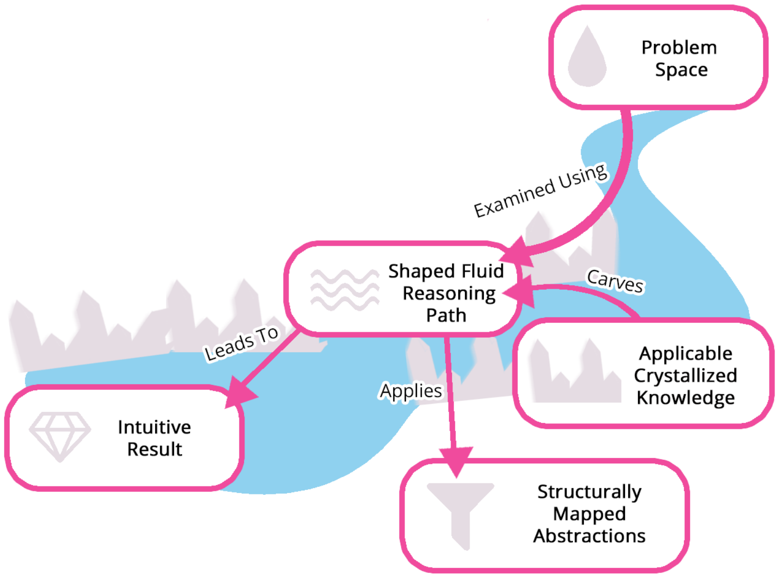

Problem Space
Examined Using
Shaped Fluid Reasoning Path
Carves
Applicable Crystallized Knowledge
Leads To
Applies
Intuitive Result
Structurally Mapped Abstractions

Figure 2.4: Diagram: proposed ideas on how novel knowledge develops into intuition

Chapter 3

Connected Shards: Analogy as a Tool

Themes of prior knowledge in mind, analogical reasoning, and novel meaning/metaphor surrounding learning have been investigated in Chapter 2. Existing knowledge is ever-present and adaptable, and our minds are predisposed to analogically reason. This leads us to exploring the use of analogy as a teaching tool. We will examine:

- the analogical nature of concepts within our discipline
- the ways in which analogy is leveraged to educate in other disciplines
- perspectives from within computer science education
- the benefits and concerns in analogy for education

Use of analogy can bring immense value to education when used appropriately. We will investigate mitigating concerns regarding the use of analogy in computing education. These concerns have plagued the exploration of analogy in computing education research. This investigation proves that the concerns around analogy are not novel to computer science, and are surmountable. This investigation will aim to bridge the gap between our discipline and others in order to positively modify the perception of analogy's place within learning computer science.

3.0.1 Why Draw Attention to Analogy?

Analogy and metaphor use has rich history across several fields in research, education, and beyond. Fields ranging from poetry and linguistics to cognitive science and user experience design have explored analogy's role in our understanding of novel information given a known source.

Many agreed upon practices in programming education utilize analogical concepts. "Worked out examples" are based in several examples having structural similarity, which the learner must analogically reason about to develop a schema of. Subgoal labeling identifies component parts to facilitate appropriate schema development. Hand traces and visualizations are meant to analogously represent the machine's processes to help learners model and reason about them. Despite the clear analogical components in these practices, the use of analogy in computer science education research is often discouraged. This investigation hopes to draw attention to the analogical nature of existing encouraged practices, and "clear the name" of using analogy in programming education.

3.1 Analogy Across STEM Disciplines

Analogy has been used across several STEM (Science, Technology, Engineering, Math) domains as a learning tool. Computer Science is itself a STEM discipline, so it is reasonable to look to other fields in this category for research and observations.

Some applications of analogy from the field of physics were already introduced in Chapter 2. Gentner and Gentner [52] developed analogies to flowing water and crowds of people as aids in learning about electricity. Their findings suggested learners used the analogies to engage with the problem space. Gentner and Gentner also observed learner difficulty when the model they were provided was ill-fitted to the given problem. Learners with the people-based model struggled on a problem where water-based reasoning was beneficial. Linder [92] observed that light can be modeled as both a wave and a particle but not as both at the same time. Both of these research investigations provide evidence of multiple source domains being a viable learning strategy, as Nersessian had posited [120]. Linder's example shows utility in multiple sources for distinct concepts within a model. Gentner and Genter in only providing a single source to each group, does not answer the question of how reasoning may have differed for learners provided with both analogies.

Analogy can be effective in leading students incrementally toward correct knowledge. Clement [18] used bridging analogies in order to progressively align student conceptions about physics phenomena. Clement further describes "anchoring ideas" as those that students believe to be true with a reasonable level of confidence [18], and that if leveraged appropriately, can be helpful [17] in progressive alignment to adapt one's mental model. Clement recognizes "brittle" anchors as those for which progressive modification changes the problem intuition for the student [18]. This causes them to believe that such alteration has destroyed some key relationship [17]. If instructors predict students believe a concept with high confidence but there is no evidence for that concept being believed with such confidence, the instructor's belief that the idea is "obvious" is in error [17]. This marks how effective classroom engagement requires continued observation and consideration of learner perception and models.

Several studies of analogy use in STEM education have shown value when the analogy is actively and critically considered. Bentley [7] assigned students to submit analogies for geologic time versus human time, followed by reviewing and discussing them in class. This research suggested that analogy composition by groups of students in order to discuss and learn is a viable strategy. Several other researchers share these observations. Haglund noted that encouraging critical thinking about analogy promotes intellectual engagement with the learning process [67]. Haglund further states that active learning activities might involve constructing an analogy or understanding a metaphor's implications within a group of peers [67]. Clement observed student benefit in evaluating analogy validity, and indicated incorporation of such activities is a constructivist teaching act [18]. Lipkens and Hayes [94] found that when composing an analogy, requiring elaboration (such as discussion) improves the learner's ability to apply the analogy. This notion of elaboration aligns with Spier et al's finding from chemistry [144]. Lower-achieving students who were asked to both create and enact (thus elaborating on) analogies for their peers out-performed their traditionally instructed counterparts [144].

Consideration of analogy in the classroom by learners is, however, best done with instructional guidance given. Zook [170] warns, that novices can lack the skill to devise their own analogies, which can impede analogy generation activities. Yerrick et al found that when left to their own devices, students may devolve into personal theories that are not conducive to the group's learning [166]. Brown notes that unguided discovery can be dangerous, and that active learning activities must be used alongside careful guidance by instructors [10]. Similarly, Duit et al found that analogous learning about chaos physics was successful, but required instructor "hints" to guide the learning activity [36]. This value from "guiding hints" parallels Gick and Holyoak's observations in prompting participants to look to the story they had read while solving a problem [60].

Instructors must also take care in considering how the learner may perceive the analogy. Glynn et al [63] cautioned that instructors may cause confusion if they utilize analogy in an unsystematic way. Zook [170] also indicated misconceptions can arise from teacher provided analogies. Taber [149] expands on this, suggesting students may not be able to interpret the instructor's intentions, or may take representations too literally [149]. Taking care with analogy design and use in the classroom is imperative. Learners must not be left unguided to reason about post-festum analogy, or similarly left to their own devices with no guidance.

Several researchers share the sentiment that reasoning by analogy is integral to innovation and science [9, 71, 94]. Clement adds that analogy can fundamentally modify one's perception of a problem space or phenomena [18]. This exemplifies the power of analogy in innovation and knowledge formation.

3.2 Analogy Within the CS Discipline

A slice of analogy's use in education across several STEM fields has been considered. Our focus now turns to analogy use in computer science education.

Research on directly on analogy use in CS Education is limited, but does exist. Chee [14] worked to implement Gentner's analogical theory in an introductory programming classroom, building instructional material surrounding a single "office" based analogy. The findings showed that those in the strong analogy condition were more adept in program comprehension and design. Chee also found that the weak analogy condition was *not* worse than no analogy at all [14]. In the weak condition, the analogy was similar but relaxed. The assistants' names were generalized and not precisely matched to some computational task, and input/output was done through one window in the office as opposed to two. This suggests that despite the pitfalls that can exist for instructor designed analogy, students were not observed to be "better off" with no analogy in contrast to a weaker one. In similar positive findings, Forišek and Steinová [45] have developed analogies for teaching the algorithmic design and properties of data structures. They found that inclusion of a valid metaphor resulted in greater student success compared with no metaphor usage Forišek and Steinová [45]. In developing pedagogical analogy for data structures similar to Forišek and Steinová [45], James [79] incorporated Black music and DJing into assignment design for a data structures course. This posed relevant problems from data structures as topical analogs, with a finding of enhanced engagement and enjoyment from learners [79]. James cites Gay's pedagogical bridge [48] as inspiration. A pedagogical bridge acts as a tool for new knowledge to be better understood through inclusion of relevant experiences [48]. The pedagogical bridge serves to connect target domain knowledge to experiential source domains from the learner's existing models. James' work finds applicability for this concept within computer science education.

Additional research in CS Education explores the communication of analogy by instructors and its effect. Sanford et al [137] explored instructional metaphors used by computer science educators. Their work aimed to better understand the metaphors used by instructors and their effectiveness in explaining the target concepts. Metaphors identified by instructors often appeared to classify concepts as physical objects: "return statements are like Harry Potter portkeys", "variables are like boxes", "pointers are like zombies" [137]. They note the need for identification of an analogy's limits. Examples based in objects like the above exemplify this: we must have an understanding of what relationships we are mapping to find relevant meaning. Alizadeh et al explored applying analogy within data science tutoring sessions [30]. They noted that 90% of the words spoken during the session were by tutors, but that when students engaged and contributed to the session, it had a positive effect on learning [30]. This provides an exemplar within computer science of the value in active engagement on an analogy's implications.

CS Education Research has also looked to compare instructional methods within tools, including analogy. Harsley et al [70] compared problem-solving value for various approaches within their computer science tutoring system. Participants using the system were presented with one of three cases: analogy, analogy and worked out example, or simply worked out example. Harsley et al indicated that analogy was found least beneficial when learners had prior knowledge of the problem space [70]. Participants within the analogy pool, given their prior knowledge, may have already possessed generalized schemas that the analogy did not work to critique. Additionally, the study design was done "in time", with the approach provided to the student alongside a problem. The solution learners provided to that problem was the success metric used — Harsley et al did not explore long-term retention and retrieval in this study. With Perkins caution toward "ritual knowledge" as a fragile knowledge example [129], additional testing of after-effects and transfer may have allowed broader insights.

Analogy is inherent in our disciplinary tools, which affects our teaching [4]. Flow charts Arawjo observes, have historic origin in engineering [4]. He highlights that their appropriation to programming attempted to help switch operators understand command inputs better. However, switch operators did not necessarily have background that allowed the flowcharts to be interpreted meaningfully, especially as they are not in Western left-to-right reading style [4]. Flowchart utility, even the prevalence of keyboards, all came from analogical reasoning in specific contexts as to the solution to a disciplinary problem. These solutions may have been pragmatic at the time, but were largely contextual [4] and now persist in the discipline as remnant structures. Learners may not (similar to the switch operators) have background or context that allows them to recognize the value of flowcharts as a result. Wozny observed that metaphoric failure occurs when the metaphor creates confusion, or its extent and applicability are unknown[165]. Krishnamurthi and Fisler remind us that we choose pedagogy, problems, languages, and notional machines [87]. With the nature of computing being steeped in metaphor, we must be mindful of the context we present our students. Analogous tools, interfaces, and syntax can otherwise become a source of confusion. We must also be careful that the internalized analogies and cultures of our field does not bind our reasoning about the possibilities of computer science education and human-computer interfaces moving forward. Even something so "ncessary" to programming as keyboards was an analogous solution to a contextual problem. Our internal perceptions of "what programming and programming education look like" should not be allowed to hold back progress in our field.

In exploring our disciplinary tools, our pedagogical tools in CS Education can reveal analogous natures as well. Carbonell [12] identifies "storing solutions" as a viable learning strategy, observing that solutions can share analogous properties to one another. These commonalities can encourage development of generalized schema through an ancestry in analogically reasoning about such solutions [12]. This fits to the value that has been observed for worked out examples in computer science education [30, 70, 140]. Similarly, subgoal and task labeling of examples, explored by researchers such as Morrison et al [117], provides explicit schema scaffolding alongside worked out examples. Hand-tracing is the action of physically representing the execution steps of a program, creating an analogy by which one might reason about execution behavior. Cunningham et al [24] explored novice applications for hand-tracing. They observed that novices tend to avoid hand-tracing, implement partial solutions, or adopt non-traditional behaviors derived from prior problem sketching contexts [24]. This showcases attempted reduction of cognitive load through a mental model's parsimonious and superstitious proprerties. Lee and Ko [91] designed an anthropomorphic compiler that posed itself as a partner to the programmer. Its suggestions and approach were framed as a co-investigator eager to understand and solve the problem, rather than the more authoritarian nature of its traditional counterpart. They found that novice programmers learned better through engagement with this personified tool [91]. This identifies both a way in which our tools might be better analogized, and also how dialectic behavior can lead to greater elaboration, understanding, and solution design. Through an interface which served as a more user-friendly analogy to the process, learning gains were found.

Other CS Education studies have explored creative expression of programming ideas, which requires analogous ideas. McDermott et al ran an experiment inviting student creativity into program development by crafting a narrative [104]. The motivation was that narrative elements become analogs that can encourage understanding and use of specific coding concepts. They identify Cooper et al [22] as integral to the idea of transforming algorithmic thinking to a storytelling context. McDermott et al observed greater student engagement with this approach due to the creative nature, but that understanding was still "fragile knowledge". They suggested further focus on mapping narrative elements to the code and problem structure was a necessity [104]. Grover et al [64] explored non-programming interactive activities with relevant narrative contexts prior to application in code. Their research found promise for such an approach, and specifically note that this approach allows for incremental abstraction of ideas [64]. Suh [148] investigates the application of comics in learning programming, both with instructor designed comics as well as comic creation activities. Such research fits both with the identified value of narrative creativity, as well as analogy generation and elaboration. Work in computer science and general STEM education has also explored developing games to increase engagement and learning of domain concepts [11, 85, 99]. Gamification requires the creation of some analogy to a target pedagogical concept. Games can increase enjoyment and engagement with regard to a topic [69, 116, 139], further adding to the value analogy use brings to STEM and computer science education.

Ghosts that Haunt CS Analogy

This breadth of analogous examples and experiments within our field might suggest analogy's value within computer science education is well-formed. However, this is not the case. Analogy — and especially cross-domain analogy — has earned a "bad reputation" within computer science education[1], despite the evidence within our discipline and across STEM disciplines of its value. Much of this appears to come from a lack of recognition that certain practices require analogical encoding or reasoning, such as worked examples, utilizing the same processes as cross-domain analogy. There is a dissonance in thinking about pedagogical approaches that require analogical reasoning, and the use of analogy itself. Many of the analogous examples and experiments indicated prior did not explore their analogous nature, highlighting the dissonance in perception. This section investigates further some evidence that may aid in understanding how the analogical nature of many CS Education practices, and cross-domain analogy use, may have evolved.

In their work investigating computer science teachers' metaphors, Sanford et al [137], noted that all metaphors break down. This is a true statement, but not an argument against analogy. When

[1]It is important at this juncture to indicate that computing education research has a heavy leaning toward CS1 research at present. As a result, findings or observations from research aimed at CS1 can end up generalized to computing education *as a whole*

analogy is grounded in elaboration activities and situated appropriately in context, we have already noted its effectiveness. Further, the citation which correlates to this observation is not in fact about all analogy: it is specifically describing a specific conversation surrounding a non-computer science analogy in which one participant notes that “the analogy breaks down” [102]. Further, Sanford et al do not identify this specifically as a weakness, but a strength if well understood, a point toward the development and critique of well-structured analogies [137]. This shows one “ghost” haunting our disciplinary thoughts on analogy — a situated context citation is extended to a general statement, becoming part of the belief system held in our discipline.

Edsger Dijkstra [31], a prominent figure in computer science, was a notably antagonistic toward analogy’s use. He noted that as our future differs from our past, words and their meaning can become ill-fit[31]. This is true, and a cultural issue in computer science with our rigid syntax while human language and culture continuously evolve. However, this is not specifically a case against analogy, and is at its core a rather extremist condemnation of polysemy in language. Dijkstra described computing as “too novel to be represented by analogy or anthropomorphization” [31]. The novelty of computing does ring of truth, as Colburn and Shute [20] identified computer science as a discipline often developing its own subject matter. This development process is novel among traditional science disciplines. However, this novelty furthers the case for analogy: in fact, for many analogies. If computer science as a phenomena cannot be accurately captured by a single analogy, then much like Linder’s light phenomena, we may need a multitude of analogies to understand and represent it. Dijkstra suggests that there is a fear in teaching “radical novelty” as itself, but this stance is not grounded in our knowledge of cognition. Our understanding of phenomena utilizes our prior notions, and if no context exists, we will develop our own, creating our own models. To teach computer science as “radical novelty” in the way Dijkstra suggests begets the assumption that students can be “blank slates” on the topic. We know students are not blank slates in their cognitive processes. They will generate their own contexts and correlations even if we try to remove them, due to analogical reasoning being part of the nature of human cognition. What’s more, while the field of technology’s metaphoric underpinnings can pose their own issues, they are still present in programming and the vocabulary our field uses. These elements cannot be removed, thus connection to prior notions and analogy cannot feasibly be erased.

Further, Dijkstra accuses analogy of infantilizing the curriculum: if one can blame a bug for “sneaking in”, then the programmer is not to blame [31]. This point is not only valid, but important to internalize: words and their connotations have meaning and can modify a culture and one’s perceptions and experiences in it. However, Dijkstra conflates linguistics and personal experience as a blame to place on analogy and anthropomorphization. Learners and experts have anthropomorphized ideas, papers, and even their own words and brains in an effort to connect to shared human experience. In stating “my brain is running all over the place” I am not anthropomorphizing my brain in order to shift blame and be coddled: I am doing so in order to draw an analog that another can understand and relate to, in order to foster camaraderie within the community of practice. A “bug sneaking in” is not so much blame passing as it is a relatable experience. Most programmers have felt like a bug just “appeared” after all their work. The surprise and dismay at finding it is relatable, with the program bug analog to an insect sneaking in creating a salient image grounded in physical experiences to express it. While it is important to consider that such language may develop into blame passing and infantilization, this is a linguistic evolution issue we must be aware of, not one exclusive to analogy. Dijkstra’s dissent toward analogy is quite frankly, lacking in cognitive relevance and often disparate from analogy or anthropomorphy being solely under fire. However, Dijkstra’s prominence within the field of computer science gives credence to his opinions — and thus, another ghost enters the discipline.

Halasz and Moran [68] describe the difficulty that can come from using analogy to explain and extend understanding of computer concepts. They note that features from the physical world analog may be irrelevant to the computing target. They do so with a salacious title that evokes Dijkstra, *Analogy Considered Harmful*[2]. Further, they evoke the novelty and uniqueness argument as a reason that concepts from the digital realm cannot leverage physical analogs. However, while citing Gentner and describing structure mapping, the authors seem to miss the concepts that drive structure mapping and systematicity. Their concerns regarding a lack of drawers in a computer file system when a filing cabinet has drawers confuses a surface feature for a relational feature. The relationships and reasoning about them allow the development of abstract schema. Surface features may aid in recall from long term memory, hence, why we may use a file cabinet as an example, given the ability to visualize it due to surface feature recall. In chapter 2, however, we noted that surface features aid in retrieval, but relationships aid in reasoning. Of course, Halasz and Moran raise a valid point that we must ensure that drawers don't appear to be relationally relevant — this can be relegated in many of the ways we have previously seen, through active engagement and elaboration. Further, they suggest instead that literary metaphor ("She moved like a gazelle") is preferable to analogical reasoning [68]. As we have already explored with our understanding of cognition, analogical reasoning is an aspect of our thought processes. Further, literary metaphor requires analogical reasoning as we have already observed. It appears Halasz and Moran believe literary metaphor is easier to process, but the same reasoning must be undergone to some degree if the metaphor is novel — which in teaching computing to a novice, it would be.

Qian and Lehman's [132] discussion of misconceptions resulting from teaching largely focuses on analogy as a primary culprit. They cite Taber [149] specifically, who wrote the following about analogy:

> "This may be either because students do not realize when such teaching devices as analogy, models, metaphors, and anthropomorphisms are being used to help make the unfamiliar familiar and so take these representations too literally [...]" [149]

We have seen the need for engagement with the analogy process, structural design, and explanation already, which can help to address this concern. This quotation can also apply to visualizations as models, which the paper later describes the effective use of. Visualization models may be arguably more susceptible to the above phenomena, as learners may come to take the representations as literally what is happening, despite them being a pedagogical tool that by nature contains abstraction (and, are in fact analogies in and of themselves). The work also cites Halasz and Moran [68], whom we just explored. This sentiment and treatment is not exclusive to this work — it is pervasive through the field. This contrast of analogy to visualizations when the cited concern exists for both showcases the ways the perceptual lens of analogy continues to haunt the field.

While not directly an analogy ghost, mental model superstitions can also affect the perception of analogy. Krishnamurthi and Fisler [87] cite Slotta and Chi [141], as well as Gupta et al [66] in suggesting it is more difficult to develop a new mental model than to update a "flawed" one. As we explored in Chapter 2, Slotta and Chi described a specific instance of naive physics conceptions and argued that in other cases adaptation was feasible and encouraged, and Gupta et al argued that even their specified case should be considered dynamic and adaptable. The sentiment of flawed being cognitively harder to revise than new development has been shared across computer science education discussions. This perception that mental models cannot adapt and evolve easily would

[2]While not cited here, this title parallel's Dijkstra's widely discussed work within the computer science field at large, *Goto Considered Harmful*

make the abstraction, re-representation, and generalization processes described previously appear nonviable, when we know they are part of human cognition and learning. Further, these sentiments suggest learning is “one shot”, and that if a learner’s model is incomplete or contains inaccuracy from the onset, we must begin entirely anew. This conception can cause wariness to approaches that do not feel like perfect initial representation. Analogy has been placed in this category of wariness within computer science education.

Even the Computing Education Handbook [42] all but dismisses analogy. The index presents no listing for “analogy” or “metaphor”. The closest listing provided is “analogical encoding”. This is described for a single paragraph, which observes “analogical encoding” as different from “instructional analogy”, but does not provide any indication of analogy as useful in computing education [117]. One can find index entries for concepts that make use of analogical reasoning or analogical representations, such as “worked examples”, “notional machines”, and “unplugged activities”. The consolidation of all of analogical processing to a single paragraph, with no indication toward any value in the use of analogy within the field’s handbook makes a clear statement of perception in its omission.

Arawjo’s notion of historical factors affecting the culture of writing code [4] appear to have extended to the research surrounding teaching as well. Problem solving from one context influences citation to other contexts, without consideration of situatedness. Prominent figures within the field and their conceptions influence and bias our future perceptions. The prominence of these figures encourages adaption to their viewpoints [95]. The cautions regarding analogy design are abstracted alongside such arguments, turning into fear surrounding their use. Communication of this fear among practitioners perpetuates the culture. Taber indicated that folk theory may be at odds with science [149]. The science suggests analogy, including cross-domain analogy, is a viable strategy with guided and engaged application.

3.3 Additional Considerations for Analogy Design

Analogy research across STEM and CS revealed valuable considerations in the design of analogy. This section highlights other relevant considerations that should be noted.

3.3.1 Cognitive

A major concern across any learning environment is cognitive load as learners explore novel ideas and problems. Both Holyoak and Richland [73] and Yuan et al [168] note several additives that can reduce cognitive load.

Holyoak and Richland specifically describe worked examples, visual representations, subgoals and gestures as valuable [73]. Yuan et al note that worked examples with annotations (such as subgoal labeling) can aid in learners ability to apply schemas. Holyoak and Richland also noted that analogy should avoid surface distraction and not require too much cognitive resource devotion to background knowledge. Yuan et al observed that connection to prior knowledge can reduce cognitive load [168]. Minimizing cognitive resources for analogy should ensure the analogy makes use of prior knowledge and experiences. Holyoak and Richland noted that their suggestions in tandem with quality analogy can maximize representation development and transfer for learners [73].

3.3.2 Context

Analogy design must take context into consideration. Forišek and Steinová note that cultural and demographic considerations, such as gender, must be taken into account with the design of analogy [45]. We may take for granted our own context, assuming the same experiences and ideas exist in the learner's context. In the same vein, good analogy design is also relevant to the learner. As such, the cultural and demographic of analogy can be a strength if it is situated around the learner's knowledge. As James [79] noted in his application of Gay's pedagogical bridging [48], culturally competent and relevant analogy can be highly engaging to the learner. The goal is not sanitizing of culture from analogy (the notion that this could be done is itself an oppressive bias). Rather, recognition of the context sensitivity allows us to be mindful of it, and leverage it as a strength. Good analogy design should consider such factors, and should work to maximize flexibility in source domain representations. An ability to be flexible with the source domain while maintaining the structure of the analogy allows for pivoting based on relevant contexts to the learner.

Gentner and Loewenstein [58] found that the way cases are presented to learners modifies perception and encoding. They found that studying one case at at a time resulted in more contextually bound representations [58], which are harder to retrieve in new situations or generalize. In working with analogy, ensuring both analogs are presented concurrently during the process is important to generalization. The presentation context can modify the represented context.

Finally, Lave et al [89] note that problem solving is not an independent act. The environment's context, identified problem, and learner context are all together shaping the problem solving context. Analogy can allow learners to close the gap of their existing knowledge and its relevance to problem spaces within the learning context, helping solutions to take shape.

3.4 Benefits in Analogical Methods

The prior sections have presented a background on researched applications of analogy across STEM and within CS. We have considered additional factors that may impact analogy design as well. Now, let us summarize the benefits that analogy in learning can provide.

3.4.1 Engagement

The act of storytelling is one of the oldest means by which information was shared across cultures. The morals of fairy tales pass cultural knowledge and wisdom through analogy. Storytelling can captivate an audience, encourage reflection and sharing, and convey key concepts. Our investigation in this research has revealed that the creativity in developing analogical narrative can be valuable and engaging to learners [22, 104, 144]. Even if the learner lacks context, storytelling can appropriately generate context while maintaining interest. Those who have engaged with fantasy or science fiction narratives know that prior experience is not a requisite to understand the story or learn from its representations and morals.

More generally, engagement as a value can be seen across analogical representations such as gamification, visualization, or culturally relevant bridging and analogy. Brown [10] noted that engagement can help students move toward recognizing intrinsic rewards in learning. Intrinsic motivation can push the learner to develop deeper understanding and interest.

While engagement does not inherently beget learning gains, it is a boon in fostering them. Haglund suggests that research on analogy should shift away from solely examining effectiveness, and instead look to recognize the value it has on learner engagement [67]. As we already know that analogy can be an effective learning strategy when applied properly, the additional engagement it may create is beneficial.

3.4.2 Pathways

We have already noted that abstraction is a key benefit of the analogical reasoning process through the development of generalized schema. Analogy's ability to bridge domains and experiences with new learning allows it to foster connection of models in mind. Gentner and Loewenstein [58] identified how analogical reasoning about experiences allowed for previous inert knowledge to become part of a schema. Such acts open new pathways in mind and allow us to apply new frames to old knowledge.

Further, the ability to create paths to ideas across domains, and the potential exploration of such pathways, is at the root of creative innovation [9, 71]. Helping learners begin to carve such paths may lead to learners who more readily develop and explore them, becoming the innovators of the future.

3.4.3 Connection

The ability for analogy to make what may seem alien or irrelevant become grounded for the learner cannot be overstated. Relevance can increase interest, engagement, and enjoyment, allowing a learner to feel connected to the topic. Learners can find themselves and what matters to their context within the material, and as such, feel more a part of the discipline.

Ryoo et al [136] noted that the programming learners they surveyed had a wide variety of passions and interests, and that their aspirations in applying pedagogical knowledge were also quite distinct. They note that educational theory rests on connecting content to relevant and understood knowledge and topics. Their work highlights that the application of concepts and ideas of interest to the student can increase motivation toward the material. Analogy can allow one way to connect source domains of interest to the learner to pedagogical content.

In James's application of Gay's pedagogical bridge [48, 79], he noted three themes in making culturally responsive bridges: connection of prior knowledge to new knowledge, use of the skills in practice, and enjoyment/interest [79]. The application of analogy allows for culturally relevant prior knowledge to connect to new learning. Utilizing analogy may help a learner better apply skills in practice, and critique of analogy can allow for generalization of necessary skills allowing them to be put into practice. Given the ability for analogy to be grounded in what is valuable and relevant to the learner, it can increase enjoyment and interest in the topic. Thus, analogy allows a method for culturally responsive bridging that is relevant to the learner. The connection of existing knowledge to new learning also can create a connection for the learner with the topic itself.

3.5 Bridging the Gap: Addressing the Concerns

While this chapter has made the case for the value of analogy, we would be remiss if we did not also identify the concerns. Here, let us explore methods and considerations that may help mitigate concerns regarding the use of analogy in the classroom.

3.5.1 Active Participation and Discussion

A major concern of analogy use was the ability of the learner to understand the analogy, as well as the fear of incorrect assumptions based on analogy. Candidate inference based in analogical reasoning is one of its strengths as a process, but can also lead to difficulty or confusion if the inference is misguided.

A key component of addressing this concern is involving the learner in the analogical process. We have seen the case for analogy generation by learners — this involvement allows them to take ownership and become invested in the reasoning as part of the process. We also observed that analogical reasoning was most effective when an instructor acted as a guide.

We've identified the concern that instructor analogies, typically didactic in nature, may lead to confusion. If analogies are being presented by the instructor, the learners must have active participation in the process. Instructional use of analogy does not need to be a pitfall, but it must be guided and dialectic. Haglund observes:

> "it is in the promotion of the critical scrutiny in challenging the analogy, attempting to apply it and recognizing when and why it breaks down that the opportunity for learning really takes place. The focus is in this sense diverted from attempts to find the perfect analogy." [67]

Allowing learners to explore the implications of others' analogies, including (especially!) their instructors, allows them to internalize and become a part of the process. Instructors can also identify engagement by probing for expected outcomes in the target domain, and how they relate to the source. Brown [10] noted that responding with a question that challenges will allow for better assessment of understanding than unchallenged answers. Sanford et al [137] already rightly identified that breakdown of all analogies occurs. If analogy is presented in the classroom, its limitations should be explored alongside students, so they engage with and internalize those representations as well. By working through the analogy with learners, they can be involved in the process.

Instructors can challenge understanding by asking about implications of action to the source: "what will happen if we do X?". This should be followed with a challenge to structural transfer by questioning the implication for the target. The ability to recognize how a candidate inference toward the source impacts the target allows for structural abstraction to be observed. Through this, we can also recognize if the learner needs more guidance or additional methods. As Friere wrote:

> "Dialogical theory requires that the world be unveiled." [46]

Analogy can allow for a dialogue with the learner and among learners. Using it dialectically alongside questions that challenge structural representations can allow for gap closing and abstraction development.

Gentner and Hoyos note that the invention to compare elements prompts pattern recognition [53]. Inviting learners to the conversation when using analogy can help promote structural alignment and consideration.

3.5.2 Structural and Contextual Relevance

We have already identified throughout this chapter that a key concern in analogy is the representation the learner takes away from its use. Greater understanding of structure mapping and systematicity [49] can allow us to develop structurally sound analogies. Such analogies should be coupled with learner participation in the process. However, the ability to identify if an analogy can appropriately convey the desired representation is a concern that must be addressed in instructional design. In the next section, we will further address ways of determining this.

It is also a concern that analogy must strike contextual balance. It must be relevant and (hopefully) interesting to the learner, but given sociocultural factors, what is relevant to one learner may not be relevant to all. Analogy design which maintains structure, but is able to isomorphically modify domains, would be of value in addressing this issue. This would reduce the problem again to one of structure so long as instructors can identify a relevant example for the learner which also possesses that structure. Domain isomorphism could also allow further abstraction, as learners are able to explore multiple domains which share the same structure, progressively aligning a more general schema.

The need for context and structure also resonates in our assessment of the applicability of analogy. The analogy "a variable is like a box" has often been discussed in computer science for its failure as an analogy due to a multitude of incorrect candidate inferences that can be drawn. The question of context thus begets another question: *in what situation was this analogy relevant?* If our introduction to the variable is to describe the entire concept as a box, then of course this has malignment, because variables are not boxes: they are variables. What problem did describing a variable as a box attempt to solve? Were we attempting to explain how variables can hold information? If so, why describe the entire structure with this analogy? The analogy and its structure is based in a specific context and misconception the learner may have. Use of the analogy should be mindfully situated to context.

This further exemplifies the case for Nersessian's multiple source domains [120]. In one situation, we may liken the structure of variable behavior to a box. In another, the variable may behave differently. The ability for mental models to interconnect and merge is a benefit of our cognition. Use of a multitude of valid source domains to describe distinct behaviors of our digital world, and engagement with the implications of those domains so that they become generalized schema in relation to the target, can allow us to develop several connected generalized models that more cohesively describe the behavior of our target as a whole.

3.5.3 Well-Formed Analogy Design

One of the major arguments against analogy has been misapplications or poorly designed analogies. Understanding how to better inform analogical design can majorly improve the quality of analogy we use, our understanding of their application, and our ability to engage learners in conversations surrounding the implications of the analogy. Clement [19] noted that validation of analogies is important. In this section, we will explore some theories surrounding the design, assessment, and validation of analogy.

Designing Analogy

Linsey et al [93] developed a method to stimulate analogy generation within business and marketing. They note the consideration that effective retrieval depends on how we originally stored information [93]. Thus, their approach, called WordTrees, expands definitions and connections of words, creating a sort of visual traversal through interconnection definitions and representations. Making explicit the connections and connotations one has to an idea, as well as those that are inert or unknown (such as adding in dictionary definitions and building from those), allows for approaches to be discovered. If one is having difficulty finding an analog to develop analogy with, an activity like this may help stimulate ideas. Linsey et al position this as a group activity. Utilizing WordTrees as an active learning exercise where groups work together toward valuable analogs may also be beneficial.

Carbonell [12], in considering artificial intelligence system design, noted the value of a means-ends analysis (MEA) in promoting an effective structural plan. Gick and Holyoak [60] later modified Carbonell's approach in order to identify states within their fortress story and their parallels to the Duncker tumor problem. They compared the story and several courses of action (presented to participants as different states of the experiment) in order to reveal the analogs to the tumor problem. They observed actions and their results, problem settings, goal states, and constraints. Identification such as this can help in designing an analogy in a situated way, and understanding key components in the analogy's design that preserve its structure.

Validating Analogy

As we saw above, Gick and Holyoak's modified MEA approach [60] allows for a form of validation within its design. In requiring explicit elaboration between the problem, goals, constraints, and solutions, the creator of the analogy must identify how the structure and relationships are preserved between their source and the target. This act allows for validation that such structure and relationships do exist.

Forbus et al [41, 44] implemented a validation model for structural analogy as part of their development of the structure mapping engine. Their approach involves identification and expression of structural correspondences, and the mapping of them between the representative elements from the domains. Where Forbus et al identify the expression of these within a programmed system, Gentner's initial work on structure mapping [49] showcased the approach by simply writing the predicate assertions. For example: *REVOLVES-AROUND(planet, sun)* exemplifies the relation in the source domain of a planet revolving around the sum. This pairs to, in Gentner's example *REVOLVES-AROUND(electron, nucleus)* from the target domain, showing that an electron also revolves around a nucleus. Structure mapping allows the identification of such correspondences and their strength in systematicity, validating the structure of an analogy.

Assessing Analogy

Of concern to educators is that their analogies are effective for learners. Several models exist for guiding analogy use in education and assessing its effectiveness.

Glynn [63] provided the Teaching-with-Analogies Model, also posited that it can lead to an effective Learning-with-Analogies Model to help learners understand how to reason and engage with analogy:

1. Introduce target

2. Cue source retrieval
3. Draw attention to relevant features of target and source
4. Map target and source similarities and correspondences
5. Indicate breakdowns in analogy
6. Draw conclusions

This model allows for a step-by-step approach to the introduction of analogy. It can also help instructors identify areas where they can increase learner engagement with the analogy by noting which steps they may turn into active learning activities.

Holyoak and Richland [73] note strategies to effective analogy use with six tips:

1. Source analogs should have a structure well known to learners
2. Compare two or more source analogs before beginning transfer to a target
3. Guide the comparison of analogs in some way
4. Identify which sources may be easier for learners to grasp and which may be more challenging. Allow for progressive alignment by moving from easy to hard.
5. Reduce cognitive load required to reason about the analogy
6. Reduce additional cognitive demands by both designing and working with analogy in a way that spotlights key relations and downplays distractors

Educators can assess if they are following these tips effectively, which can lead to better usage of analogy within the classroom.

Zeitoun [169] also offers a nine stage model for teaching with scientific analogies in "The General Model of Analogy Teaching" (GMAT):

1. Measure learner traits relevant to analogical learning
2. Assess prior knowledge of the target domain
3. Analyze target domain material
4. Determine analogy appropriateness
5. Identify analogy characteristics
6. Choose presentation and engagement approach in classroom
7. Evaluate results of analogy use
8. Revise and Iterate

This model provides the instructor a more holistic view of the classroom and learners, which may aid in analogy design and classroom methods used. It also introduces a cycle for instructors to continue the process of fine-tuning their use of analogy within the classroom.

Chapter 4

Opus: The OPAL Evaluation Tool

As we have seen in previous chapters, analogy is a core cognitive process and can provide learning benefits. I developed the **Outlining Programming Analogy Layout** (OPAL) tool in order to help instructors better design and assess the structure of their analogies. OPAL allows for computing analogies to be developed more precisely and with structure in mind. This will hopefully aid in elevating the conversations surrounding analogy in computing education.

This chapter explores the components of OPAL and how they benefit the design of analogy. Issues that arose in analogy design through use of OPAL are analyzed, exemplifying how OPAL encourages cognizance of potential pitfalls. OPAL analogies are shown not only to be transformable to other structural design methods, but also to encourage the potential for consideration of additional source domains for a developed analogy. The usability of OPAL is considered through multiple methods of engaging with other educators and their analogies.

4.1 Understanding OPAL

OPAL is intended to promote critical analysis and documentation of analogy design. Striking a balance between accessibility and rigor, my aim with OPAL is to facilitate stronger design of structured analogy in an easy-to-use way.

A well-structured analogy should be formulated with:

- identification of the target and source domains
- recognition of what concept the analogy is meant to communicate
- knowledge of the relations to be represented by the domains
- comparable relation and entity representation across both domains

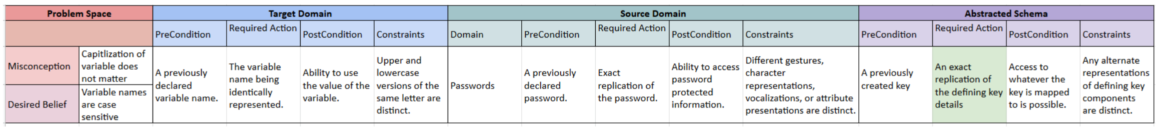
To guide inclusion of the above considerations, OPAL has four main components:

1. Identification of analogy context
2. Target domain (programming) procedure
3. Source domain (analogous) procedure

4. General structure relating both domains' procedures

OPAL's components are used to exemplify a **procedure** occurring across each domain. OPAL promotes approaching analogy by considering *actions* and their *results* across the domains. This design is intended to encourage stronger development of systematic relations: *"If X action results in Y in the target domain, what action achieves a comparable result in the source domain?"*

This procedural focus is fitting for programming, and can address concerns surrounding entity-focused analogy. Instead of simply making the entity-based claim "a variable is like a box", one must reason about "what specific process am I attempting to communicate where a variable and a box behave similarly?" This reasoning narrows the focus and requires relations be present in the exemplified process. Programmers complete actions by typing code, and those actions produce some result. Framing analogy based on the process of action-result is situated to the act of programming and promotes relational consideration.



Problem Space		Target Domain				Source Domain					Abstracted Schema			
		PreCondition	Required Action	PostCondition	Constraints	Domain	PreCondition	Required Action	PostCondition	Constraints	PreCondition	Required Action	PostCondition	Constraints
Misconception	Capitlization of variable does not matter	A previously declared variable name.	The variable name being identically represented.	Ability to use the value of the variable.	Upper and lowercase versions of the same letter are distinct.	Passwords	A previously declared password.	Exact replication of the password.	Ability to access password protected information.	Different gestures, character representations, vocalizations, or attribute presentations are distinct.	A previously created key	An exact replication of the defining key details	Access to whatever the key is mapped to is possible.	Any alternate representations of defining key components are distinct.
Desired Belief	Variable names are case sensitive													

Figure 4.1: Screenshot of a completed OPAL analysis within a spreadsheet entry

Table 4.1 shows the same example as the image, expanded for easier readability in print.

Table 4.1: Variable Names are Case Sensitive - Passwords (Appendix A.1)

Identification of Analogy Context	
Misconception	Capitalization of variable does not matter
Desired Knowledge	Variable names are case sensitive
Exploration of Target Domain (Programming) Procedure	
Precondition	A previously declared variable name.
Required Action	The variable name must be replicated exactly.
Postcondition	Ability to use the value of the variable.
Constraints	Upper and lowercase versions of the same letter are distinct.
Exploration of Source Domain Procedure	
Domain	Passwords
Precondition	A previously declared password.
Required Action	The password must be replicated exactly.
Postcondition	Ability to access password protected information.
Constraints	Different gestures, character representations, vocalizations, or attribute presentations are distinct.
Analysis of Common Structural Elements	
Precondition	A previously created key.
Required Action	An exact replication of the defining key details.
Postcondition	Access to whatever the key is mapped to.
Constraints	Any alternate representations of defining key components are distinct.

4.1.1 Identification of Analogy Context

Analogy is used to communicate information within a specific context. This OPAL component requires identification of what situations the analogy is relevant for. An analogy surrounding array indexing does not provide value if the learner's difficulty relates to an array's fixed size. Recognizing the analogy's intended purpose helps to ensure it will be used appropriately.

The "Desired Knowledge" represents the idea the analogy is designed to communicate — what we hope the learner internalizes from it. Is the analogy designed to show correct behavior? Does it replicate incorrect behavior to show the learner why this behavior is incorrect? The "Misconception" portion is optional, but useful when the analogy can target some misunderstanding. If the analogy may be used to encourage mental model adaptation regarding a specific misconception, this field should be used to identify the misconception it can apply to.

Reflecting on the intended takeaway of the analogy ensures the design of the analogy aligns with this objective. This should minimize misapplication of the analogy, which will enhance its learning value as a result.

4.1.2 Target Domain Procedure

For an analogy to be effective, the source material must have a component and structural relationship to the target material. OPAL was designed for a programming context, so the identified target domain is expected to exemplify some programming concept.

The target domain's procedure must align with the analogy context. If the procedure and indicated context do not match, the designed analogy will not be applicable to this context. This can result from an incorrect assumption of appropriate context, or the design process diverging from the goal. Ensuring alignment between the context and the procedure described promotes contextual grounding of the analogy.

The Target Domain Procedure contains four components:

1. Precondition
2. Required (or Mistaken) Action
3. Postcondition
4. Constraints

Precondition

In the Precondition, components that are required for the analogy's context are identified. If an analogy targets array element access, one must have an array to be able to reason about accessing its elements. This constitutes a precondition: *What is the "setup" required to reason about this idea?*

The precondition serves two major purposes. First, it allows us an additional check to ensure the analogy is used in the appropriate context. As the precondition elements *must* be present for the procedure to continue, we have another verification that the analogy will be used in appropriate contexts. Second, the elements present in the precondition provide a basis for the relationships and entities of the analogy. If the precondition indicates an array, an entity that represents an array must also appear in the source domain's precondition. The precondition allows for "stage setup" in the analogy's design.

Required (or Mistaken) Action

The Required Action identifies what steps must be taken to arrive at the procedure's result. This can also be referred to as the Mistaken Action, if the analogy is used to emphasize how the wrong action leads to the wrong result.

In an analogy which communicates accessing an element in an array, the required action should describe what a learner must do to access an element. Given that the precondition holds (so long as an array exists), this is the action to take. In Java, this would be naming the array variable, followed by indicating the element's index. Specifying the action to be taken given the precondition forms at least one relation our analogy must convey. Whatever entity exists in our source must also be able to conduct an action that maps to this action.

Postcondition

The Postcondition describes the observable outcomes after the required action is completed. If the required action of accessing an array element is executed, the postcondition indicates that the value stored at that element is retrieved. If a mistaken action was taken, the post condition should reflect the incorrect state that results from it.

Constraints

Constraints are optional, but can allow clarifications that may impact interpretation of the procedure's result if not identified. In the variable name analogy that Table 4.1 showed, it was important to indicate how different *forms* of the same character were distinct — 'B' and 'b' behave differently. This consideration may not have been fully recognized through the action sequence, but is necessary for the procedure to behave correctly. Inclusion of necessary constraints adds consideration of additional entities or relations that may be needed in developing the analogy. This component is not necessary in all analogies, but aids in the rigor and structure of those it applies to.

4.1.3 Source Domain Procedure

The source domain is the area we are drawing knowledge from in order to reason about programming. Source and target domains should have entities and relations that appropriately map to each other in well-formed analogies. The source domain contains the same four components of the target domain as a result.

One additional value in the source domain is indicating *what* source is being considered. While our target analogies always relate to programming in OPAL, our sources may come from many different fields and contexts. Identifying the source domain of the analogy can allow for one to assess the analogy's relevance to the learner given their prior knowledge.

The source domain must follow the same procedure of identifying precondition, required action, postcondition, and constraints (if any existed in the target domain). In completing these steps, one must ensure entities and relations map to the ideas presented in the target. If the precondition in our target requires an array, the target specified a key entity. We must ensure our source also contains a key entity that maps to an array. Whatever required action we conduct on our array must then have a mappable action that is conducted on the key entity from the source.

Breaking down the procedure to four component parts allows consideration of entities and relations at each step of the analogy design. It is difficult to assert an analogy is well-formed if identified relationships from the target do not have a mapped equivalent in the source. Each step provides a "check", leading to whole source and target domain designs that can be compared more critically.

4.1.4 General Structure Relating Both Domains' Procedures

A final step that allows assertion of the analogy's form is abstracting the general relational structure it presents. This general structure encapsulates *how* reasoning about the source domain might generalize into relevant understanding for the target domain. For the instructor, this step asserts the abstracted schema analogical reasoning should promote for the learner.

This step can be the most difficult to consider, but also one of the most valuable. The difficulty lies in explaining generally the entities and their relations. This must be done in a way that is broad enough that both source and target fit with this general structure, but not so broad that the entities and relations become nebulous and vague.

The value in constructing the generalized structure is not only validation of a well-formed analogy, but potential to recognize additional applicable source domains. If an analogy likens an array to "a row of mailboxes mounted to a post", this may generalize as "an ordered collection of items with a fixed size". This language can allow consideration of additional source domains: *what else is an ordered collection of items with a fixed size? Can those things also structurally fit this action sequence?* One may generate additional source ideas through the use of this generalized language — but identifying this language can be tricky.

This step may not be imperative for all instructors if an additional structural check is unneeded or consideration of more source domains is not desired. For instructors who are first working with OPAL, this portion may be hard to consider. It may be better left as a later exercise when one becomes more comfortable with using the tool.

"Impact" Points

The presented examples of OPAL shown featured green highlighted boxes. These were used to indicate perceived "impact" points in the procedure. These points appear to be where the "punch" in communicating the idea occurs when one is communicating the analogy. Different analogies may have different points at which a key idea or relation is conveyed. This observation is entirely optional, but may prove interesting in observing the design and structure of one's analogies.

4.1.5 Impacts of Using OPAL in Instructional Design

OPAL forces instructors to interrogate the components and relationships within their analogies, and directly compare them to the programming target. This can be an eye-opening process. One may realize, for example, that the entities of their analogy make sense, but process relationships have not been adequately realized.

OPAL is also flexible: while target-to-source generation is explored in "Understanding OPAL", one *can* design using source-to-target. If an instructor identifies a source first (perhaps one they already had in mind or were considering), they must investigate a procedure with that source and identify how this maps to a target scenario. They may recognize a need to rework aspects to fit the desired objectives, or consider if the source provides value for ideas they wish to communicate. OPAL is functional in helping develop analogies and in critiquing previously designed or considered ones.

The generalized structure portion of OPAL adds value in allowing consideration of multiple source domains. With a well-reasoned structure, further source domains can be developed that also fit this structure. As learners have diverse prior knowledge and experiences, the ability to "pivot" source domains and still produce a well-formed analogy is valuable. Using multiple source domains can encourage further abstraction by learners as well. The utility of multiple sources can be pedagogically valuable even if a previously considered source is already applicable to the learner.

OPAL forces contextualization of analogy, requiring that the analogy fits an idea to communicate, and that the procedures described fit this context. This helps instructors identify appropriate applications of their analogies. When the analogy is contextually relevant, conveying it can provide value to the learner and minimizes confusion about the analogy's intention.

Worth further investigation is how OPAL promotes a "storytelling" structure. Using story and narrative as a vehicle for analogy can encourage conjuring of appropriate context even if one lacks prior knowledge. OPAL's design encourages one to describe the analogy through a process with a beginning, middle, and end — paralleling general story structure. This may result in context generation through how it encourages one to convey the analogy. Appropriate context generation and engagement with narrative can promote additional reasoning and transfer during the learning process.

4.2 Applying OPAL to CS1 Misconceptions

To test the design of OPAL, I utilized misconceptions that were gathered from the Fall 2018 and Spring 2019 semesters as a basis for developing analogies. Misconceptions that appeared to possess enough entities and relations to form a reasonable action sequence were selected as candidates for this process. Some previously utilized analogies from the CS1 course were also used with OPAL in this process. This allowed verification of its flexibility for both design and critique. In an initial assessment of OPAL on CS1 misconceptions, 55 analogies were developed. These initial analogies can be found in Appendix A. Their state during initial development is preserved in this Appendix, with minor grammatical and clarity modifications.

4.2.1 Considerations During Initial Testing

Many of the misconceptions chosen had no source domain consideration prior to working with OPAL. The ability to use OPAL to guide reasonable analogy development was showcased in working with these misconceptions. One such analogy is the comparison of integer division to cooking, shown in Table 4.2.

Table 4.2: Division and Floating Points - Cooking (Appendix A.9)

Identification of Analogy Context	
Misconception	Setting the result to double will stop integer division truncation
Desired Knowledge	Floating point must be introduced to the division for floating point results to be remembered
Exploration of Target Domain (Programming) Procedure	
Precondition	Two integers being divided.
Required Action	Division introduces a double.
Postcondition	The result can be a double with fractional elements retained.
Constraints	Double must occur during division not only as the result.
Exploration of Source Domain Procedure	
Domain	Cooking.
Precondition	A recipe containing wheat.
Required Action	Gluten allergies require ingredient inclusion during steps.
Postcondition	The result modifies the recipe to be gluten-free.
Constraints	Ingredients need to be changed during recipe preparation not after.
Analysis of Common Structural Elements	
Precondition	Action utilizing one type of information.
Required Action	A different type of information is introduced to the steps of operation.
Postcondition	The different information can be assimilated into the result.
Constraints	Information must be introduced during operational steps, not after.

Previously used analogies or source domains were also critiqued through OPAL. One example is comparison of the object creation process to a factory, and that constructors allow for “additional setup” in the form of instance variables. This is shown in Table 4.3.

Table 4.3: Instance Variable Setup - Manufacturing (Appendix A.32)

Identification of Analogy Context	
Misconception	Created constructor does not do anything
Desired Knowledge	Constructor allows for instance variables to be set up for object
Exploration of Target Domain (Programming) Procedure	
Precondition	A class template.
Required Action	Use constructor to prepare object's instance variables.
Postcondition	Object is ready for use when initalized.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Manufacturing.
Precondition	A factory line is used to create something.
Required Action	Before it is shipped final details are prepared so it is ready to use.
Postcondition	The item is ready to be used at the end of creation.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A template for creating something.
Required Action	Use the final step of the creation to get the thing ready for use.
Postcondition	Thing that is created is fully prepared to be used.
Constraints	None.

One objective in applying OPAL was exploring its utility in considering multiple source domains. Thus a broad range of source domains were considered with multiple source domains used for some of the concepts. Since a learner must have knowledge of the source domain in order to reason about it, a prime consideration in the choice of source domains was the likelihood that our learners would be familiar with it.

4.2.2 Analogy Critiques and Revision

Using OPAL to develop initial analogies also encouraged further critique and analysis. During the initial application, I was still working to better understand what was required for structural relation in programming analogies. Appendix B houses alterations to some of the initial analogies using the critical perspective OPAL affords. This section explores each of these cases and how using OPAL to analyze them prompted revision.

Lost, Still Unfound

Table 4.5 revises Table 4.4, changing the domain from Gathering Mail to attending a White Elephant party. This revision was made after analysis of a significant relational oversight in 4.4.

Table 4.4: Scanner Storage - Mail Sorting (Appendix A.3)

Identification of Analogy Context	
Misconception	Scanners store inputs to variables automatically
Desired Knowledge	Obtained values must be stored immediately or they are lost
Exploration of Target Domain (Programming) Procedure	
Precondition	An initialized Scanner.
Required Action	Call next methods to obtain input.
Postcondition	Scanner has current input at time of next call.
Constraints	Information is only held at time of next call, and discarded after.
Exploration of Source Domain Procedure	
Domain	Sorting Mail
Precondition	A mailbox with mail in it.
Required Action	Visit mailbox to get mail.
Postcondition	You have the mail in your hand after visiting box.
Constraints	Without storing the mail somewhere, you are likely to lose it.
Analysis of Common Structural Elements	
Precondition	A means of obtaining input information.
Required Action	Obtain information one piece at a time.
Postcondition	Current piece of information is shown when obtained.
Constraints	Information shown is lost unless stored.

Table 4.5: Revised 4.4: Scanner Storage - White Elephant (Appendix B.1)

Identification of Analogy Context	
Misconception	Scanners stores input to variables automatically
Desired Knowledge	Obtained values must be stored immediately or they are lost
Exploration of Target Domain (Programming) Procedure	
Precondition	An initialized Scanner.
Required Action	Call next methods to obtain input.
Postcondition	Scanner has current input at time of next call.
Constraints	Information is only held at time of next call, and discarded after.
Exploration of Source Domain Procedure	
Domain	White Elephant Exchange
Precondition	A pile of presents.
Required Action	Do some action to get a new present.
Postcondition	You have the one present you obtain after the action.
Constraints	If another action happens, you will always get a new present unless you have acted in a way that allows you to ”hold” your present.
Analysis of Common Structural Elements	
Precondition	A means of obtaining information.
Required Action	Obtain information one piece at a time.
Postcondition	Current piece of information is shown when obtained.
Constraints	Information shown is lost unless stored.

The analogy originally compared Scanner not being able to return past values to a person not attending to their mail and losing it. It is highly unlikely for one to lose their mail right after retrieving it. Further, lost mail can be recovered. This is not at all similar to the relationship that a Scanner has with the information it scans. If Scanner is requested to retrieve information, that must be used/stored immediately, or it cannot be retrieved. It was obvious in considering the practical implications of how one might think about losing mail that this analogy needed revision. It could give learners new misconceptions, such as that a "search for previous values" method exists for Scanners.

A white elephant gift exchange was instead chosen. This better describes the idea of only being able to hold onto one item at a time, and if one does not choose to "leave the game" to hold onto the present (storing the information outside the Scanner), they are subject to whatever actions come next and can lose the previous item. This comes closer than the mail analogy as it better targets "one item" at a time being considered, as well as that the action of the programmer affects the outcome (keep or toss), but it does present some new problems to further consider.

The first issue is sociocultural. White elephant gift exchanges are a holiday season novelty in the United States, primarily in specific family groups and industry cultures. This makes the white elephant a weak analogy for many students, who may never have attended such an event and thus have no source knowledge.

The second issue is that a white elephant exchange still does not fully realize the relationship Scanner and input values have. If one loses a gift they're interested in during a white elephant, they can still "keep track" of the gift as the exchange progresses, attempting to retrieve it. The person still has knowledge of what they held before. In contrast, Scanner behaves with only knowledge of the current "gift" it holds. If an action changes the held gift, Scanner does not have the ability to remember what it held prior. Humans can recall prior values, Scanner cannot. While the white elephant gift exchange gets *closer* than mail does, it still does not accurately convey the relationship fully.

My continued difficulty in pinpointing this design may have been due to attempting to embody the learner within the process. The learner's ability to remember their past actions and desires raises obstacles in paralleling a Scanner. A better design might consider an inanimate object that provides information based on environmental input. Being inanimate, one cannot ask it what previous information it held. Something like a weather vane could suit this purpose: at any time we can "request" information from it, but it only points to the current value of wind direction, and it cannot be asked to recall prior values once the wind direction has changed.

Designs like this highlight the ways programmatic processes can seem at odds with the physical world. The ways *we* interact with and understand the world differ from the ways specific concepts may perform. We don't instantly forget the last thing we just did (usually), but a Scanner has no construct to represent previously stored values. Critiquing processes in this way gives empathy as to why they can be difficult for novice programmers to understand. This encourages identifying better ways to articulate these ideas.

Coming to Common Ground

Table 4.7 revises Table 4.6, with the modification of domain from Printers to Assistants.

Table 4.6: Multiple Scanners - Office Printers (Appendix A.4)

Identification of Analogy Context	
Misconception	Multiple scanner creation for single input source
Desired Knowledge	Only one scanner per location is necessary
Exploration of Target Domain (Programming) Procedure	
Precondition	An initialized Scanner.
Required Action	Invoke next methods on the same scanner to obtain input.
Postcondition	Scanner will obtain further input.
Constraints	Additional Scanners initialized to the same location will cause issues.
Exploration of Source Domain Procedure	
Domain	Office Printers
Precondition	A copy machine in an office workroom.
Required Action	Send items to print to machine remotely using commands.
Postcondition	The machine obtains commands and prints jobs.
Constraints	Additional copy machines in the same workroom may cause confusion as to which machine has your printout.
Analysis of Common Structural Elements	
Precondition	A means of obtaining input from a location.
Required Action	Continuing to utilize the same input means.
Postcondition	Obtain further input from that location.
Constraints	Multiple obtainers at same location causes confusion.

Table 4.7: Revised 4.6: Multiple Scanners - Office Assistants (Appendix B.2)

Identification of Analogy Context	
Misconception	Multiple scanner creation for single input source
Desired Knowledge	Only one scanner per location is necessary
Exploration of Target Domain (Programming) Procedure	
Precondition	An initialized Scanner.
Required Action	Invoke next methods on the same scanner to obtain input.
Postcondition	Scanner will obtain further input.
Constraints	Additional Scanners initialized to the same location may cause resource access issues.
Exploration of Source Domain Procedure	
Domain	Office Assistants
Precondition	Assistant in an office.
Required Action	Assistant is asked to complete a task using the copy room's resources.
Postcondition	The assistant does the requested job.
Constraints	If multiple assistants are asked to do similar tasks in the same room, they may complete them, but they may also run out of resources, become confused, or repeat/miss steps trying to divvy up work.
Analysis of Common Structural Elements	
Precondition	A means of obtaining input from a location.
Required Action	Continuing to utilize the same input means.
Postcondition	Obtain further input from that location.
Constraints	Multiple obtainers at same location can cause confusion.

In the initial analogy development, physical machines in a room were used, with someone sending work to a specific machine and becoming confused as to the location of the completed work. In Java, if a learner creates multiple Scanners that all access console input the learner may experience issues when using these Scanners throughout their program. Explaining this problem to students is an especially tricky one, because they may not always see the issues depending on their implementation. Resource management and stream control are prime culprits, but scenarios invoking these may not always occur unless specific actions are invoked on the common stream (such as closing one of the Scanners), or may only crop up within specific runtime environments. Thus, an analogy conveying the idea of confusion can create an appropriate structural relationship to the problem space that abstracts well.

While the original analogy does convey the concept of confusion, as well as confusion based on location (Scanners pointing to one input source, machines located in one room), the confusion is resolvable in a way that it is not within programming. If one labelled the machines (named the Scanners, which one must do to create a Scanner variable), there should be no confusion. Even if the machines were unlabeled or the label could not be seen, a person can simply walk to each machine and assess if the completed work is there. The confusion is with the programmer in this analogy, and the programmer can "solve" the confusion through variable names and conditionals. The analogy required exemplification of confusion among what was portrayed as machines.

Further, multi-Scanner errors can cause exceptional circumstances that stop the program from running - a confused person in a copy room does not mimic the situation, as they are not going to suddenly stop working due to confusion. Thus, the idea of a job being unable to be completed needed to be introduced. Inability to complete in the real world can often occur due to a lack of resources making it physically impossible to do so - and as resource management is a concern among multiple Scanners, this also fits well with the analogy.

The analogy was revised to incorporate multiple people, as now the co-located elements are able to more accurately express the relationship of confusion. Further, they may confuse each other with improper directions or movement of resources such as paper and scissors. Finally, the assistants may not coordinate on their resource usage, and in fact, deplete the resources available, becoming unable to complete a job appropriately. All of these relationships make this analogy much more viable in explaining the potential issues with multiple scanners, even if learners have no issues (all the assistants didn't get any each other's way). Further, this analogy strongly associates the idea of a physical resource being part of the problem, which can help in abstraction to resource management within the computer.

Fast or Fresh?

Table 4.9 showcases the switching of domain for Table 4.8 from ordering Fast Food to making a sandwich at home.

Table 4.8: Primitive Wrappers - Ordering Fast Food (Appendix A.6)

Identification of Analogy Context	
Misconception	Primitives and their wrapper classes are identical
Desired Knowledge	Primitives and reference types are distinct
Exploration of Target Domain (Programming) Procedure	
Precondition	A situation where a primitive that has a wrapper class is needed.
Required Action	If only value is needed, primitive is used.
Postcondition	Value is accessed simply.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Ordering Fast Food.
Precondition	A burger and a burger wrapper.
Required Action	If eating the burger immediately, wrapper is unnecessary.
Postcondition	Burger can be accessed without wrapper overhead.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	Element with value and element representing element with value.
Required Action	When only value is needed element with value is used.
Postcondition	Simplest access to value is carried out.
Constraints	None.

Table 4.9: Revised 4.8: Primitive Wrappers - Making Sandwiches (Appendix B.3)

Identification of Analogy Context	
Misconception	Primitives and their wrapper classes are identical
Desired Knowledge	Primitives and reference types are distinct
Exploration of Target Domain (Programming) Procedure	
Precondition	A situation where a primitive that has a wrapper class is needed.
Required Action	If only value is needed, primitive is used.
Postcondition	Value is accessed simply.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Making Food
Precondition	A sandwich and a lunch box.
Required Action	If you are going to eat the sandwich now, you don't need to store it in the lunchbox.
Postcondition	Easy access to the sandwich without additional steps or overhead.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	Element with value, and an element that is representative of the value element.
Required Action	When only value is needed, element with value is used .
Postcondition	Simplest access to value is carried out.
Constraints	None.

The original analogy suggested that using a wrapper class when a primitive value is desired is like an unnecessary wrapper on fast food you plan to eat immediately. The change resulted in a much healthier analogy, but this perspective was not the influence for change. The core concept can be lost entirely in the fast food analogy. Fast food at almost all franchises is delivered wrapped whether you order it for dine-in or takeout. This does not suit the analogy, which was meant to convey that adding more complexity when less will do is unnecessary. Thus, the analogy was shifted to someone cooking at home and making a decision to then eat or store a sandwich. This lends to the immediacy of primitive variables and value access. While it's easy enough to take a sandwich out of a lunchbox, if one had just made the sandwich themselves to eat, it is absurd to put it in one just to take it back out. The positioning of the person (programmer) making a choice based on utility, and one that showcases the value primitives have in lacking overhead shows through well with this shift.

Location, Location, Location

Table 4.11 focuses on revising Table 4.10, one of the first analogies to be identified as requiring further analysis and also one of the more difficult revisions to make.

Table 4.10: Void Return - Calling a Friend (Appendix A.33)

Identification of Analogy Context	
Misconception	Void methods cannot do actions
Desired Knowledge	Not returning does not mean nothing happens
Exploration of Target Domain (Programming) Procedure	
Precondition	A void method.
Required Action	Modify a reference type defined outside method.
Postcondition	Changes to reference type are seen beyond void method scope.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Calling a Friend.
Precondition	Calling a friend who is long distance to chat.
Required Action	Sharing stories and future plans with your friend.
Postcondition	Friends action perspective and knowledge are altered even though nothing physically changed.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A set of instructions that return no physical results.
Required Action	Actions occur that modify results nonphysically.
Postcondition	Change occurs despite no result being returned.
Constraints	None.

Table 4.11: Revised 4.10: Void Actions - Safety Deposit Box (Appendix B.4)

Identification of Analogy Context	
Misconception	Void methods cannot do actions
Desired Knowledge	Not returning does not mean nothing happens
Exploration of Target Domain (Programming) Procedure	
Precondition	A void method passed a reference argument.
Required Action	Modify aspects of passed reference type's information.
Postcondition	Changes are seen beyond method's scope.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Safety Deposit Box
Precondition	You and your friend share a safety deposit box, and you have placed an item for them in it.
Required Action	Your friend accesses the box and locates the item.
Postcondition	Changes your friend makes to the item before returning it to the box for you will be there when you later check the box.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	Being given access to a location with items.
Required Action	Change aspects of the items at location.
Postcondition	Changes are visible to anyone accessing that location.
Constraints	None.

The original analogy compared the modification of reference type variables within a method to calling a friend, who can influence how you're feeling despite not being physically present.

One preliminary difficulty in developing this analogy was chosing how to approach it. The indicated misconception could use multiple approaches to modify it, such as printing information or calling other actions. The desired knowledge for this analogy was chosen as reference type implications are often difficult for learners to understand — thus, this analogy could also serve a purpose for other misconceptions surrounding methods and reference types. In hindsight, greater specificity in the misconception choice may have made it easier to approach this analogy's design from the onset.

Learners initially confront the idea that methods in Java are pass by value — if "x" equals 3 and x is sent to a method, only x's *value* of 3 is sent, not x itself. The value for reference types is a reference to their memory location. If y is a reference type, it "exists" somewhere in memory, and where it *exists* is the passed value, not *what* exists there. When this location-based value is sent, changes made to what is stored at that memory location will be reflected outside the method. After beginning to understand primitives and their functionality, reference types appear to throw a curveball to learners by upending some notions they may have developed. Thus, using this example to focus on reference types felt like a worthy endeavor for this work and for learners. However, embodying the idea of reference types proved to be challenging in the real world.

The idea of location being imperative was present in the original analogy, but not invoked correctly. Location was characterized through distance: in Table 4.10, the friend is a long distance away. This was intended to showcase the inability to "change location", correlating to reassignment of the alias, which would modify the procedure. However, it only conveys a sense of distance and the ability to alter something at distance. This confuses the idea of location in memory with scope location, where "somewhere else" is another method. The ability to represent the modification that occurred is also not conveyed well. While our friend may have acquired new knowledge, we do not "see" knowledge — and we literally do not see over a voice call. With reference types, modifications to their elements are visible beyond the scope, so long as the memory location is maintained across scopes. This observable behavior is important to the procedure, but feels abstract in the original analogy.

With revision to a safety deposit box, the idea of visible change and physical location is better conveyed. A safety deposit box is a place that holds things, which draws a direct parallel to a reference type accessing a location in memory that holds information. The safety deposit box also allows an extensible approach in understanding aliasing: it is an immobile location, but the owner may claim a new safety deposit box instead — modifying the location they look at to another place. Even when moving items from one box to another, the box itself is not copied. Items must be physically moved or replicated if the new safety deposit box should be identical to the first.

By being an immovable location holding things, one must visit the location to view the things. This further allows for the analogy: that others who are able to visit that location and change things there may do so, and when the same location is visited again, the changes made will still be there.

"Visitation" encapsulating the core analogy action draws a better corollary to reference types. A location in memory is "visited" and the information stored there is interacted with. "Calling a friend" simply does not embody this idea well. Creating an appropriate analogy required interrogating some of the novelty reference types present. The original analogy skirts the idea, using an abstract route to explain the "otherness" of reference types. In concretely assessing the structure of what occurs in the machine, stronger relational designs become apparent.

In critiquing Table A.33 and trying to better understand what felt off about the analogy, my explanations of the topic even without analogy became stronger. My approach to the topic changed as I realized that learners may not fully understand reference types not *just* because their new, but because of implications in how they behave differently. I can confidently say that my approach to this topic has become richer – and hopefully more edifying for learners — after reflection on and revision of this analogy.

Backwards and Forwards: They all End up the Same

Table 4.13 revises Table 4.12, but also misses the mark in designing a truly valuable analogy for this misconception. In both the first analysis and the revised version, the same question surrounded their design: *does there exist a scenario where a "while" situation cannot easily be changed to an "until"?*

Table 4.12: For Loop Condition - Crossing the Street (Appendix A.45)

Identification of Analogy Context	
Misconception	Loop condition is “go until”
Desired Knowledge	Loop condition is “go while”
Exploration of Target Domain (Programming) Procedure	
Precondition	A set of instructions contained within a loop construct.
Required Action	Creating a condition that continues repetition WHILE it is TRUE.
Postcondition	A loop that repeats the steps inside UNTIL the condition is FALSE.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Crossing the Street.
Precondition	You start at one end of the street.
Required Action	While you have not reached the other end you stay alert.
Postcondition	Once you are NOT crossing the street you stop being constantly alert.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A scenario where steps will be repeated.
Required Action	Condtions to continue doing the steps under.
Postcondition	Steps being repeated appropriately while conditions hold true.
Constraints	None.

Table 4.13: Revised 4.12: For Loop Condition - Setting the Table (Appendix B.5)

Identification of Analogy Context	
Misconception	For loop condition is “go until”
Desired Knowledge	For loop condition is “go while”
Exploration of Target Domain (Programming) Procedure	
Precondition	A for loop.
Required Action	A starting value and condition that is true while between the first and last value.
Postcondition	The loop executes the correct number of times.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Setting the Table.
Precondition	A circular table for a meal with many guests that must be set.
Required Action	Choosing a spot to begin setting the table, and setting each place while there are still utensils and spots remaining.
Postcondition	The correct number of places are set at the table.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A scenario where steps will be repeated a certain number of times.
Required Action	Where to begin, and a check for repeating steps that allows for repetition while it stays true.
Postcondition	The steps are repeated the correct number of times.
Constraints	None.

In all my analysis on this analogy, I'm still left saying "no" to the question surrounding this analogy — highlighting my difficulty with it. The analogies presented showcase the "go while" idea, but they do not remove the ability of a learner to design a "go until" condition. *While you are crossing the street, you look both ways* is equivalent to *until you have crossed the street, you look both ways.* Similarly, *while you have utensils and spaces left, you set the table* can easily be expressed as *until there are no utensils and spaces left, you set the table.*

Even while editing this dissertation, the question still lingers. One suggestion was a container we wish to fill with water. While the container is not full, we add water. If we add water until full, the container starting full can introduce a problem. However, this consideration changes the *structure* of the problem, targeting the "always run once" nature of "do while" versus the "check first" of "while". This is not inherently targeting "until". The analogous case would be "go until the container is full". The loop would not be entered as a result of this condition, because the container is already full. This is the same behavior we would expect of the "while" — thus, the ability to express both "while" and "until" with this suggestion persists.

Each procedure plays out the "while" variant but can easily be reconfigured to the "until". If a learner has misconceptions surrounding this, the analogy does nothing to present a compelling reason as to why a "for" loop's condition is a "while" condition. The analogy only *expresses* the "while" condition — the learner has not seen why the "until" is incorrect, and can still consider the example with this mindset. In this case, the analogy does not appear capable of accurately compelling any change in the learner's beliefs, because the opposite (which is what the learner *already* believes in this case) is inherent in the structure.

The real difficulty in dispelling this misconception appears to be a linguistic barrier. Learners can often express "while" conditions in isolation. When using a "for" loop, however, learners often recognize the case to "stop" first, gravitating focus toward the idea of "until". Further, the learner can develop tunnel vision and become unable to consider the design of the complementary "while" even once they *believe* have acknowledged this change must occur. My attempts in designing this analogy were unable to appear structurally viable in dispelling this misconception. However, they did help clarify the misconception's insidious nature through continued reflection. Designing "while" conditions does not help when the "until" condition can be easily considered. When a learner is stuck not on recognizing a need to convert their loop condition, but *how* to convert an until to a while, this "second head" is left unattended. Perhaps an analogy surrounding the process of *transforming* "until" to "while" may help tackle the issue more meaningfully.

Hitting too Close to Home

Tables B.6, B.7, and B.8 all describe analogies that did not have major issues with their source domains, which are presented in Tables A.51, A.52, and A.53 respectively.

However, their source domains posed a significant issue at the time of use in the classroom. The original source domains referenced a pandemic, lockdown, and family illness. These were all developed before the COVID-19 pandemic, which — apart from the terrible loss of life that occurred on a global scale — caused mass stay-at-home efforts and isolation in order to curb its spread. Presenting the analogies in their original form during the campus closure would have certainly been "relevant" to our learner's context, but callous and insensitive at best, and cruel at worst.

These analogies were thus reworked in order to be less associated with the crisis. To rapidly be able to rework these analogies revealed the ease with which analogies could be shifted to a new domain when using OPAL. One can see the similar structures and relationships within the compared analogies, but the ability to pivot source domain allowed for socioculturally relevant (or in this case, non-antagonizing) analogies to still be well-formed.

To exemplify this source shift, Table 4.15 shows the transformation done to Table 4.14.

Table 4.14: Try Container - Pandemic (Appendix A.51)

Identification of Analogy Context	
Misconception	Only first instance of possible exception needs to be in a try block
Desired Knowledge	Any code that could be affected by the exception must be in a try block
Exploration of Target Domain (Programming) Procedure	
Precondition	Code that can throw an exception.
Required Action	All code that the exception can affect is in a try block.
Postcondition	The code should appropriately handle exceptions.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Pandemic.
Precondition	A contagious disease is noted.
Required Action	Any thing that could spread the disease is quarentined and sanitized for safety.
Postcondition	The contagious disease should be handled and not spread.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A potentially dangerous situation.
Required Action	Anything that could be affected has cautionary measures enacted.
Postcondition	The situation should be appropriately handled.
Constraints	None.

Table 4.15: Revised 4.14: Try Container - Nuclear Radiation (Appendix B.6)

Identification of Analogy Context	
Misconception	Only first instance of possible exception needs to be in a try block
Desired Knowledge	Any code that could be affected by the exception must be in a try block
Exploration of Target Domain (Programming) Procedure	
Precondition	Code that can throw an exception.
Required Action	All code that the exception can affect is in a try block.
Postcondition	The code should appropriately handle exceptions.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Nuclear Radiation.
Precondition	An unusual substance that emits toxic nuclear radiation is located.
Required Action	Any objects it touched that could also have radiation are contained.
Postcondition	The radiation will not spread and the situation is handled.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A potentially dangerous situation.
Required Action	Anything that could be affected has cautionary measures enacted.
Postcondition	The situation should be appropriately handled.
Constraints	None.

4.3 Consideration of OPAL's Usability

To ensure OPAL promotes good analogy design, additional methods of using it for design and analysis were explored. These showcase that OPAL "stacks up" to existing structural analogy design, and that other individuals can utilize it as a design tool as well.

4.3.1 Comparison Against Existing Methods

OPAL components can be transformed to fit with existing methods for identifying analogy structure. This transformation capability positions the value of OPAL in designing structured analogy. Following these transformations, the distinct value that OPAL provides in comparison will be highlighted.

I will use the OPAL analogy from Table 4.16 for my comparison to existing methods.

Table 4.16: Reference Types Create Aliases - Pets (Appendix A.40)

Identification of Analogy Context	
Misconception	Setting two arrays equal duplicates the contents of one to the other
Desired Knowledge	Assigning an array to point to another reference is aliasing but there is only one array with two names
Exploration of Target Domain (Programming) Procedure	
Precondition	Two distinct arrays.
Required Action	One array is set equal to another.
Postcondition	Both array names are now referring to the same array.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Pets.
Precondition	Two dogs have collars with their distinct homes.
Required Action	One dog is adopted into the other's home.
Postcondition	Both dogs collars now reference the same home.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	Two value names referencing distinct locations.
Required Action	One name is assigned to another name's location.
Postcondition	Both names are now referring to the same location.
Constraints	None.

Means Ends Analysis

Utilized by Carbonell [12] in exploring artificial intelligence design, this method was adopted and modified by Gick and Holyoak [60] to compare their story-based analogy to the problem space. OPAL and the Means-Ends Analysis both describe the structure of an analogy's design, thus OPAL can be compared to this method.

Given the OPAL analysis from Table 4.16, the developed analogy was translated to a Means-Ends Analysis in the style of Gick and Holyoak in Table 4.17.

Structure Mapping Predicate Grammar

As described by Forbus et al And Gentner [41, 44, 49], the relations between entities in an analogy can be described using this grammar. As it describes predicate-relations in designating the structure of analogy design, OPAL can be compared to this method.

Given the OPAL analysis in Table 4.16, the developed analogy was translated to a Structure Mapping Predicate Grammar in the style of Forbus et al and Gentner:

Target Domain

1. CONTAINS(code, array-1)
2. CONTAINS(code, array-2)

Table 4.17: OPAL Analogy Applied to MEA

Problem Statement	**Programming Problem**	**Analogy Form**
Problem Setting	Code has two arrays. Each array's name references its memory location. The arrays exist at different memory locations.	Neighborhood has two dogs. Each dog has its own collar indicating its home. The dogs live in different homes.
Desired Goal	Both arrays are names for the same memory location.	Both collars are references to the same home.
Problem Constraints	None	None
Solution	Array one is assigned to array two's memory location.	Dog one is moved to dog two's home.
Resulting Goal State	Array one and array two both name the same location.	Dog one and Dog two's collars both indicate the same home.

3. POSSESS(array-X, array-name)
4. REFERENCES(array-name, memory-location)
5. CAUSE(ASSIGN-TO(array-1,array2),
   EQUAL(array-1.memory-location, array-2.memory-location))
6. INDICATES(array-1.name, array-2.memory-location)
7. INDICATES(array-2.name, array-2.memory-location)

Source Domain

1. CONTAINS(neighborhood, dog-1)
2. CONTAINS(neighborhood, dog-2)
3. POSSESS(dog, dog-collar)
4. REFERENCES (dog-collar, home)
5. CAUSE (ASSIGN-TO(dog-1, dog-2), EQUAL (dog-1.home, dog-2.home))
6. INDICATES(dog-1.collar, dog-2.home)
7. INDICATES(dog-2.collar, dog-2.home)

OPAL's Distinction From Existing Methods

The prior examples showcased that OPAL analogies can be transformed to existing structural analogy methods, but OPAL also provides its own distinct value.

OPAL more greatly encourages scaffolding in the design process. The structure mapping grammar validates relational schema and systematicity, but it does not promote *how to design or realize it.* The Means-Ends Analysis encourages parallel comparison and thus more greatly encourages design approaches. It does not, however, incorporate an additional "check" as to the intended general structure. This removes one additional validation check, and also lowers the ability for multiple source domains to be identified, a value OPAL can promote.

The order of operation in the MEA is also more geared toward showing the comparative design, where OPAL's element order is meant to encourage the design act itself. The desired goal in the MEA parallels the desired knowledge, but this is much deeper in the MEA. If one were attempting to complete the MEA "top down", they may get partway through only to realize they have not appropriately contextualized the analogy they are working to form. OPAL's order promotes continual checks and validations of structure through the design process.

4.3.2 Usability with Other Instructor's Ideas

OPAL is a tool designed for instructors to critically analyze their analogy design. This section explores other instructor's analogies and use of OPAL in assessing its usability.

CS1 Instructional Team OPAL Workshop

Our CS1 instructional team includes lecturers, graduate lab instructors, and undergraduate lab assistants. During an instructional team meeting in Spring 2020, a group workshop was conducted on using OPAL. During that sememster, the course instructional team consisted of one lecturer (myself), one graduate student lab instructor teaching both lab sections, and six undergraduate lab assistants who were enrolled in upper-level courses, with three lab assistants being assigned to each section. The instructional team presented analogy ideas from their work with CS1 students in the labs, and we worked together to develop these analogies within OPAL. Appendix C contains these analogies.

Table 4.19 shows a revision of Table 4.18. This revision was done with the team's input after they indicated they did not feel confident with the original analogy. Through brainstorming we decided that nametags would provide a better analogy.

Table 4.18: Nonstatic Implicit - Board Games (Appendix A.28)

Identification of Analogy Context	
Misconception	The implicit parameter is named somewhere within the method
Desired Knowledge	By making the method nonstatic the implicit parameter is required when called
Exploration of Target Domain (Programming) Procedure	
Precondition	An instance method.
Required Action	An object calls the instance method.
Postcondition	In calling the method the object doing it is known.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Board Games.
Precondition	A piece on the board requires a player to move it.
Required Action	A player must move the piece.
Postcondition	By moving the piece the player whose piece that is is known.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	An action which requires something to do it.
Required Action	A thing must do the action.
Postcondition	By doing the action the thing doing it is known.
Constraints	None.

Table 4.19: Revised 4.18: Nonstatic Implicit - Nametags (Appendix C.1)

Identification of Analogy Context	
Misconception	The implicit parameter is named somewhere within the method
Desired Knowledge	By making the method nonstatic the implicit parameter is required when called
Exploration of Target Domain (Programming) Procedure	
Precondition	An instance method.
Required Action	An object calls the instance method.
Postcondition	In calling the method the object doing it is known.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Nametags.
Precondition	An employee is needed to do some action in a store.
Required Action	The employee does the action and you are assisted.
Postcondition	By looking at their nametag you know who it was that helped you.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	An action which requires something to do it.
Required Action	A thing must do the action.
Postcondition	By doing the action the thing doing it is known.
Constraints	None.

Tables C.2 and C.3 were both proposed by the same assistant, and showed how common structure can inform multiple domain design. This assistant indicated using both cars and a classroom to explain the idea of the "has-a" relationship being applicable to objects. These both contained the core theme of a complex element "having" many simpler elements — thus, we were able to recognize in completing this how a common relational structure can extend across domains.

Table 4.20 elicited a very interesting discussion and consideration for OPAL's design. Every application of OPAL that I had designed was to showcase the "correct" procedure or consideration when programming. The assistant proposing this analogy used it exemplify to learners *why* the incorrect method behaved incorrectly, by relating it to an action they are familiar with. Our resolution to this to show "corrected" action was the addition of the constraint, while maintaining the design of the assistant's analogy in pre, required, and post.

Table 4.20: Alias vs Parameter Locations - Discord (Appendix C.4)

Identification of Analogy Context	
Misconception	Modifying a parameter's aliasing affects the original argument sent
Desired Knowledge	The alias is not the same as the parameter
Exploration of Target Domain (Programming) Procedure	
Precondition	Having a reference type variable in one scope and passing it to another scope.
Required Action	Changing the value of the variable within the other scope.
Postcondition	Changes to the value aren't seen outside the scope.
Constraints	If you don't connect any aliases created in the other scope to the first scope then those can't be seen outside the one scope.
Exploration of Source Domain Procedure	
Domain	Discord.
Precondition	Having a Discord account and being a member of a server.
Required Action	Changing your nickname within a server.
Postcondition	Name change is only seen in the server not the account name.
Constraints	If you don't track all of your server names under your account name people in other servers won't know the differnt names you have.
Analysis of Common Structural Elements	
Precondition	The location of an item is replicated from one context to another.
Required Action	The location in the replicated context is changed to a different location.
Postcondition	The location the original context looked at is not changed.
Constraints	If information is not relayed between contexts, location redirection is not tracked.

Given the teaching context, it certainly may not be necessary for OPAL to show the behavior that "does what the learner wants". There is certainly value in helping a learner understand why the wrong approach is wrong, by relating it to something they can understand, in order to recognize that is not the behavior they want. Thus, this case was very eye-opening for OPAL's use. Clearly, OPAL still allows analogies of this "contrapositive" style to be represented. Constraints can be added to "correct" the behavior to the "positive" interpretation. However, it also highlighted an oversight in my original analogy design, which was consideration of the value that such cases might have.

Instructor Generated Analogies, Explored in OPAL

In gathering data and observations from the lab, many instructional team members indicated distinct analogies they utilized that differed from the analogy cases they were provided that week. Here, I work to situate these examples within the OPAL framework, based on the context they provided and the known topic context for that week.

This further validates OPAL in showing its ability to give structure to analogous cases I did not personally develop.

Analogies created using the notes instructors provided while collecting data in the labs can be found in Appendix D. These analogies were developed using only the context provided in the "Details Left on Data Slip" section. This may mean they are not the *exact* analogy used by the instructional team members — especially as the misconception or belief the instructor was targeting is not always apparent.

An example of using OPAL on an indicated analogy surrounding returning a value versus printing a value is shown in Table 4.21.

Table 4.21: Returning and Printing - Ordering Pizza (Appendix D.11)

Identification of Analogy Context	
Misconception	Returning and printing are the same thing
Desired Knowledge	Returning allows a value to be used outside the method, while printing simply displays it
Exploration of Target Domain (Programming) Procedure	
Precondition	A value to send back from a method.
Required Action	You can print information about the value, but only in returning it can another method have and use it.
Postcondition	The calling method has the value after it is returned.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Ordering Pizza.
Precondition	A pizza to give to a customer.
Required Action	You can describe that the pizza is ready, but only giving it to the customer allows them to have and use it.
Postcondition	The customer has the pizza after you deliver it to them.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	An entity which must change domains.
Required Action	Actions describing the entity do not allow it to change domain, only physically being sent to the correct domain works.
Postcondition	The new domain now has the entity after it has been sent.
Constraints	None.
Details Left on Data Slip	
Topic Week and Indicated Concepts	6; Return
Notes	pizza driver just announces order instead of giving you the pizza

These examples showcase the ability for OPAL to allow structural design based on a brief description of an analogy provided by others. This shows the utility of OPAL in not only assessing one's own analogy, but developing a conception of the structure of others. Further, it helps indicate OPAL can be utilized beyond my own conception of appropriate domains or processes.

Feedback on Using OPAL from Non-CS1 Instructors

To provide further validation, additional course instructors were asked to leverage the framework based on concepts from their courses. The instructors were asked to give feedback on using the tool.

Another CS1 instructor was requested for feedback on the tool, but was unable to provide any feedback due to time constraints. An instructor of the follow on course to our CS1 course, Instructor 2, provided 3 completed examples and indicated beginning to work on another example from their course. The concurrent computing instructor, Instructor 1, provided 6 examples.

The analogies these instructors created as part of this process can be found in Appendix E. Feedback comments made by the instructors are also included alongside each OPAL analogy. Additional comments from the instructors are also provided in the Appendix.

Observations on the feedback from each instructor are summarized below.

Instructor One

- Noted difficulty in designing some desired constraints
- Suggested that outlandish constraints would be tenuous
- Revealed in a discussion with me difficulty in capturing entire higher-level concepts with a single OPAL entry. After suggesting decomposing the concept into specific processes, the instructor utilized this technique.
- Indicated a desire to exemplify "why was wrong is wrong" in one analogy, not just correct procedures
- Observed that sometimes there is no misconception, only Desired Knowledge to convey
- Left the general structure of one entry blank, but with a strong enough relational analogy that I was able to suggest a functional general structure.
- Recognized a limitation in using chairs from the Dining Philosophers Problem [138] to describe countdown semaphores
- Felt the template helped them formalize designed analogies

Tables 4.22 and 4.23 showcase Instructor one's decomposition of one higher-level concept, thread waiting, into two component processes.

Table 4.22: Thread Waiting - Checkout (Customer POV) (Appendix E.1)

Identification of Analogy Context	
Misconception	A thread can always wait on a CV without conditional constraint
Desired Knowledge	As CV signals are lost if not threads are waiting, conditional contraints must be used
Exploration of Target Domain (Programming) Procedure	
Precondition	The thread wishes to wait for permission to take an action.
Required Action	The thread checks if permission has already been granted; if not the thread indicates that it wants permission and waits; if so the thread consumes permission and continues.
Postcondition	The thread only acts when permission has been granted.
Constraints	Mutual exclusive access to the monitor (structural limitation).
Exploration of Source Domain Procedure	
Domain	Checkout.
Precondition	Want to purchase items and needs a clerk available.
Required Action	Check to see if clerk is currently available; if not the person queues; if so the person starts checking out and occupies the clerk.
Postcondition	The person is only able to check out when the clerk is available.
Constraints	??? This is a really difficult constraint that I'm passing up nailing down right now.
Analysis of Common Structural Elements	
Precondition	The procedure requires a resource.
Required Action	The procedure must wait while the resource is not available.
Postcondition	The procedure completes its task with ownership of the resource.
Constraints	??? This is a really difficult constraint that I'm passing up nailing down right now.
Provided Notes	
Notes	This is a major problem that students have to tackle, and I do think I solve it with analogy. However, expressing that "larger" analogous process succinctly is challenging. Am I perhaps trying to phrase too many aspects of the problem at once? Would this work better if I decomposed this into a set of relationships

Table 4.23: Thread Waiting - Checkout (Clerk POV) (Appendix E.2)

Identification of Analogy Context	
Misconception	A thread can always wait on a CV without conditional constraint
Desired Knowledge	As CV signals are lost if not threads are waiting, conditional contraints must be used
Exploration of Target Domain (Programming) Procedure	
Precondition	The thread wishes to grant permission for another thread to take action.
Required Action	The thread checks if threads are already waiting for permission; if not the thread indicates that permission has been granted; if so the thread signals a thread waiting for permission.
Postcondition	The thread grants permission either indirectly (through state) or directly (through signalling).
Constraints	Mutual exclusive access to the monitor (structural limitation).
Exploration of Source Domain Procedure	
Domain	Checkout.
Precondition	Need to assist customers in checking out purchases.
Required Action	Check to see if customers are currently waiting; if not the clerk indicates that they are available; if so the clerk starts assisting the customer.
Postcondition	The clerk is only able to check out when a customer is waiting.
Constraints	??? This is a really difficult constraint that I'm passing up nailing down right now.
Analysis of Common Structural Elements	
Precondition	The procedure provides a resource.
Required Action	The procedure offers up the resource if no procedure is waiting for it.
Postcondition	The procedure facilitates another's task by giving ownership of a resource.
Constraints	??? This is a really difficult constraint that I'm passing up nailing down right now.
Provided Notes	
Notes	To continue on, I decomposed my initial attempt into two separate perspectives. Both are addressing the same problem and the same misconception but they frame the problem from two different actors/threads. The core misconception relies on the intersection between 4 different branches of control flow between those two different actors/threads.

Below is my suggested completion for the blank general structure of Table E.6.

- **Precondition** One or more entities wish to share a resource exclusively among their group
- **Required Action** The first entity claims the resource, and the last entity to complete use releases the resource
- **Post Condition** In claiming the resource, it cannot be used by any outsiders until every entity in the group has completed use
- **Constraints** None

Instructor Two

- Showcased quite a bit more difficulty in using OPAL
- General structure incorporated the student's misconception into the steps, rather than targeting it through the process. This is likely due to poor training on my part, as I provided OPAL with only examples and a blank worksheet to obtain feedback.
- Evidence suggests they may have worked "left to right" in the spreadsheet. The original spreadsheet had the relational structure to the left followed by target, then source. Working in this order could certainly present hurdles.
- Indicated that one of the analogies does not appear to translate well to text, but that the analogy promoted dialogue that lead to understanding with the learner. Using OPAL should promote engagement and critique of an analogy's design in promoting understanding. The "shape" of the analogy may have changed through the conversation, which may have been the reason the instructor struggled to articulate it.

Chapter 5

Tesserae: Considering Analogy and Engagement

Engagement with the learning process is a powerful factor in the adaptation and development of mental models. Analogy has the ability to more greatly encourage learner engagement. This chapter explores some considerations for capitalizing on analogy's engagement benefits.

5.1 Physical Analogical Representations

Activities and demonstrations provide physical representations, engaging the learner with their interactive nature. Integration of physical simulation or demonstration in computer science education is not novel. Initiatives like CS Unplugged [153] are devoted to designing physical representations for computing concepts. Activities such as these are analogical in nature. The demonstration is used to promote recognition of some learning concept while also fostering engagement.

Research into embodiment [142] also explores the use of physical gesture and movement while explaining programming, and how such actions can aid in learning of programming concepts. Embodiment is similarly analogous — the gesture or movement is meant to represent and exemplify some concept that is being learned about.

The ability to "visualize" through an example or assignment that feels grounded in real-world understanding can similarly promote engagement. Engaging ideas from the "real world" that learners can model and visualize promotes the use of their "real world" knowledge as a source domain to solve the problem.

Within my CS1 course, I employed several physical analogical demonstrations in lecture. I also developed new assignments that encouraged real-world visualization and analogical reasoning. The rest of this section presents a description of notable demonstrations and assignments that made use of physical analogical representations. These provided interesting ways to utilize analogy in the in order to classroom to promote engagement.

Peanut Butter and Jelly

This activity was used prior to my start in lecturing the course, and I have continued incorporating it. During the first lecture, the instructor lays out materials to create a peanut butter and jelly sandwich [1], asking the class for help in making it by requesting the steps. As learners provide steps,

[1] As a health and accessibility consideration, it may be beneficial to instead choose a more hypoallergenic nut butter

the instructor "does what they said", but often not in the way they expect. One common first step is the suggestion that the instructor "open the bread". The instructor then, in the style of the Incredible Hulk, rips open the bag two-armed from the center, sending pieces flying everywhere. Shocked, the learners begin working to formulate their instructions more precisely to achieve the sandwich making goals. If learners begin to become a bit mischievous by suggesting clearly inappropriate actions, the instructor can suggest an inability to carry out such steps to help learners refocus on the task.

After time has passed or something attempting to resemble a sandwich has been completed, discussion is held with the class. This discussion investigates what went wrong, and how we could fix it. Learners are introduced to the concept of an algorithm. They explore why the approach wasn't an algorithm, and what could have allowed the process to better fit this definition.

This activity early in the semester catches learners off-guard, intriguing them on what is usually "syllabus day". This also suggests to them this class might not be like others. I have several experiences seeing the impact conducting this activity has. I have apparently been nicknamed "Peanut Butter and Jelly" on exams several *weeks* later. I have watched learners pull out phones to film the spectacle, eager to share what they were experiencing. I have even observed past students smile knowingly as they see me walking down the hall with a grocery bag on the first day of class. These observations highlight the interest and engagement this experience has for learners.

TRex-pectations versus Reality

A small in-class activity during the objects topic helps learners first recognize the difference between classes and objects. First, learners discuss with the peers around them how they would "build" a T-Rex. What properties would they need to capture? What actions should their T-Rex do? What properties does the T-Rex need to design those actions?

Class discussion is conducted about properties and actions each group identified. Some groups show a focus on specific ideas, while others try to think of general concepts. The groups present a variety of ideas — with some being similar and others very different. The instructor also shares some of their own ideas. They then indicate that everyone had different ideas, but all of them reflected actions and properties of a T-Rex. This helps demonstrate that planning an object's design doesn't require a perfect representation, but should include what needs to be modelled for the problem space being considered.

Learners are then asked if these descriptions of T-Rexes *are* T-Rexes. The learners observe that no, of course they aren't — they are descriptions. This is used to teach the concept that a class describes functionality and properties, but is not "the thing". A T-Rex toy is then pulled out, noting that this is an *actual* representation of a T-Rex (due to the difficulty in procuring a real T-Rex for the demonstration). If the toy is imagined to be made based on the properties and actions described earlier, the toy would be a realization of the learners' descriptions — an object. This demonstration helps learners being understanding how to plan the design of a class, as well as the difference between objects and classes.

Rubber Duck Factory

Another demonstration is used to engage learners with the concept of objects. Two pre-written whiteboards are brought to class, along with a bag containing several rubber ducks with distinctive properties.

The first whiteboard details properties a rubber duck can have, and actions a rubber duck can do. The board is labeled "How to Build: Duck" at the top. The whiteboard's contents are explained to learners (especially as those in the back may be unable to read it) and they are asked what it represents. Learners identify that the whiteboard is like a class — a "template" for ducks. Calling back to the T-Rex activity, the whiteboard is physically held up as the instructor asks "is *this* a duck?" resulting in a "no" from learners.

second whiteboard is then revealed, showcasing different ways to "order" the creation of a duck, with a blank chart to indicate detail about created ducks. The possible ways of "ordering" a duck's creation are explained, and learners are asked what the ability to customize the way their duck is created is like. They observe the parallel to a constructor.

The activity of duck-building is then worked through. Different constructor "builds" determine the properties desired of the duck (the properties of ducks the instructor has available are indicated as options to learners). The instantiation of the desired duck is written out on the board in Java code. Then the corresponding rubber duck is pulled from the bag, noting it as the object that was created from the instantiation. Its properties are added to the created ducks chart. Additional ducks are created, some with different properties, and others with identical properties.

When identical ducks are created, learners are asked if the two ducks are the *same* duck, which they assert is not true. This leads to a discussion on reference equality versus property equality. Calling methods on the ducks is also conducted by determining an action from the class template and working out the appropriate code statement. "Simulating" the duck doing that action follows, such as quacking at the class. This allows learners to tangibly engage in the process of object instantiation and use.

Re-Duck-Rection

The duck friends are brought back again to help better understand the implications of reference types and aliasing. As the ducks have already been used to represent objects, and objects are reference types, this demonstration allows further connection of ideas.

Ducks are named and the implications of assignment statements between ducks are worked through. As a class, we consider the implications of statements such as "jonathon = quackingsworth" — Jonathon the duck is now equal to Quackingsworth the duck.

The first incorporation of this demonstration resulted in ducks being quite literally thrown across the room when they were re-assigned, often falling behind something. Learners were asked if we could reasonably retrieve the original information about Jonathon that had been thrown, now that Jonathon was another name for Quackingsworth. The physical act of "chucking the duck" prompted the idea that no, that information has been lost.

While the original exercise was memorable, it lacked some nuance required to deeply understand aliasing. Later demonstrations used sticky notes attached to the Ducks with their names. When a Duck was reassigned, its sticky note moved to the Duck it was assigned to. If a Duck had no sticky notes, learners were asked if there was any way to "get back" to that Duck through assignments. Without a name, there was nothing to assign the Duck to, so the answer was no — this Duck's information was lost. This allowed learners to physically see the concept that aliasing is "different names for the same thing" by seeing a multitude of sticky notes on a single Duck. Further, it made physical the *implication* of aliasing: that the variable name is now correlated with another location.

Exceptional Eraser Tossing

A physical demonstration using whiteboard erasers helped exemplify the exception handling process. The keywords in Java for this process are "throw" and "catch", which added value to this activity by using the keyword's inherent metaphoric leaning.

The instructor asked for two learners' help with the demonstration. Once two learners have volunteered, the instructor states they will take on the role of a method throwing exceptions, holding up an eraser to symbolize the exception. The learners will be exception handlers, and are then told what type of exceptions they catch. The instructor then proceeds with the activity by calling out what type of exception the eraser is and throwing it. The learners catch it if they were allowed to, or let it drop if they weren't. After each throw and subsequent catch (or lack thereof), the class is asked if they agreed with the outcome and why, allowing discussion on the simulations correctness and the ability to resolve any mistakes. Discussion is typically strongest on hierarchical exceptions. I might present the eraser as a "FileNotFound" Exception, telling the learner they could only catch "IO" Exceptions, or vice versa. Such examples prompted discussion on how the hierarchy of exceptions worked, and which cases fit in specific hierarchies.

Having two learners present for the demonstration allowed handling "chains" to be demonstrated as well. After the initial demonstration, the learners' role had complexity added: they were now both handlers *and* the methods containing them. In this portion, the learners are informed if they call another method, as well as how they handle exceptions. The instructor might be a method that is called from Learner A's method. Learner A may be called from Learner B's method, and handle exception X by throwing it. Learner B catches and handles exception X. In this example the instructor identifies the eraser as Exception X, and tosses it to Learner A, who should immediately toss it to their caller, B, who can successfully catch it. This "hot potato" example lets learners see how exceptions are "tossed up" the execution stack.

A final demonstration exemplifies why it is silly to have a method header state an exception will be thrown, only to proceed to catch the same exception. The instructor describes themselves as the method, and the eraser as the stated exception. They note the exception has occurred, and acknowledge they can "catch" it, but also that they've said to throw it, continuing to toss it in the air, catch it, and toss it. This "self-juggling" illustrates to students why using both concepts is superfluous for the same exception on the same method.

Bunnies, Bees, Bears Running Through Trees

Another analogous representation we use is not "physical", but grounded in learner understanding of the physical world, and analogically reasoning about it as a result. In our CS1 curricula, we explore "multi-object interactions" to better understand ways objects interact, and how this can be used to model simulation systems. We work through several simulations in lecture, and learners complete a simulation in lab that they later modify in their homework.

First some of the ways objects might interact are shown — objects using other objects as parameters, modifying those objects with method calls, and so on. A simple simulation of a "Jurassic Park" is shown, where T-Rex and Brontosaurus objects exist in a park. The park increments days to show the passage of time, and output on the life, death, and conflict within the park is shown. The code for the simulation is explored, and discussion is had on what portions of the code do. Conditional logic calling methods the learners have not seen yet is observed, but learners are able to understand the intention with the method call names and object-based design. The instructor notes that nothing in the simulation uses concepts the learners have not already explored in class. This helps learners recognize how the simple concepts they have learned can build to complex systems. Time allowing, aspects of the simulation are experimented with to simulate different scenarios. One demonstration had a backward conditional that prompted the accidental growth of a 90-million-ton brontosaurus who kept thriving on seemingly endless foliage in the park. This was amusing for learners and inspired engagement in finding the bug that explained this phenomena.

Further small simulation projects explored the relationships of dependency, aggregation, and association. In the dependency simulation, there is a Forest, which contains an instantiated Bear and Hiker object. The class discusses what dependencies exist — the Bear is able to chase the Hiker, but the Hiker has no methods indicating knowledge of the Bear. For aggregation, there are Dragon, Nest, and Treasure classes. Each Dragon has an array of Treasure, and each Nest has a single Dragon. This allows exploration on two forms of aggregation, as describing the treasure when the only access point is the nest requires obtaining information about the nest's dragon first. In the association simulation, there is an outline of a Fish and an Aquarium class. As a group, the class discusses which approach makes more sense — is it was more reasonable for a Fish to have an Aquarium it lived in, or an Aquarium to have the Fish that lived in it? This helps learners understand association is useful in planning and understanding a relationship, but the choosing that relationship's form impacts the final design of the code.

In lab activities, learners were provided a Produce and Garden class. They were tasked with building a Bunny class, then simulating Bunnies eating Produce in the Garden. A homework problem required modification of Produce to represent flowers rather than vegetables, and to build a Bumblebee object that pollinated the Garden. This required learners to design systems similar to the simulations seen in class, and to see how existing classes can be re-purposed in modelling new scenarios.

These simulations and modelling assignments were designed specifically to elicit strong analogous representations in mind. The learner can visualize an unaware Hiker being chased by a Bear, or a Bunny nibbling its way through a garden. This allowed salient, engaging ways to introduce learners to the power of object-oriented systems.

5.2 Fostering Engagement with Analogy

Analogy use promotes additional learning value by encouraging engagement with the material. As the previous section showed, this value in analogy can be further exemplified through unique application of analogy. The use of analogy can invite two major engagement factors in order to promote active reasoning in learners: topical interest/intrigue, and humor.

5.2.1 Topical Interest

When a topic is interesting to learners, they are more likely to consider it worthy of effort and engagement. Gay's pedagogical bridge [48] suggests topics from a learner's prior knowledge as a "bridge" to new material. James's [79] application of this bridge identified three themes for culturally responsive bridging: connections to prior knowledge, use of skills in practice, and enjoyment/interest. Selection of an analogy's source domain allows for the choice of relevant and interesting learner-centered domains, provided a well-formed analogy within that domain is able to be identified.

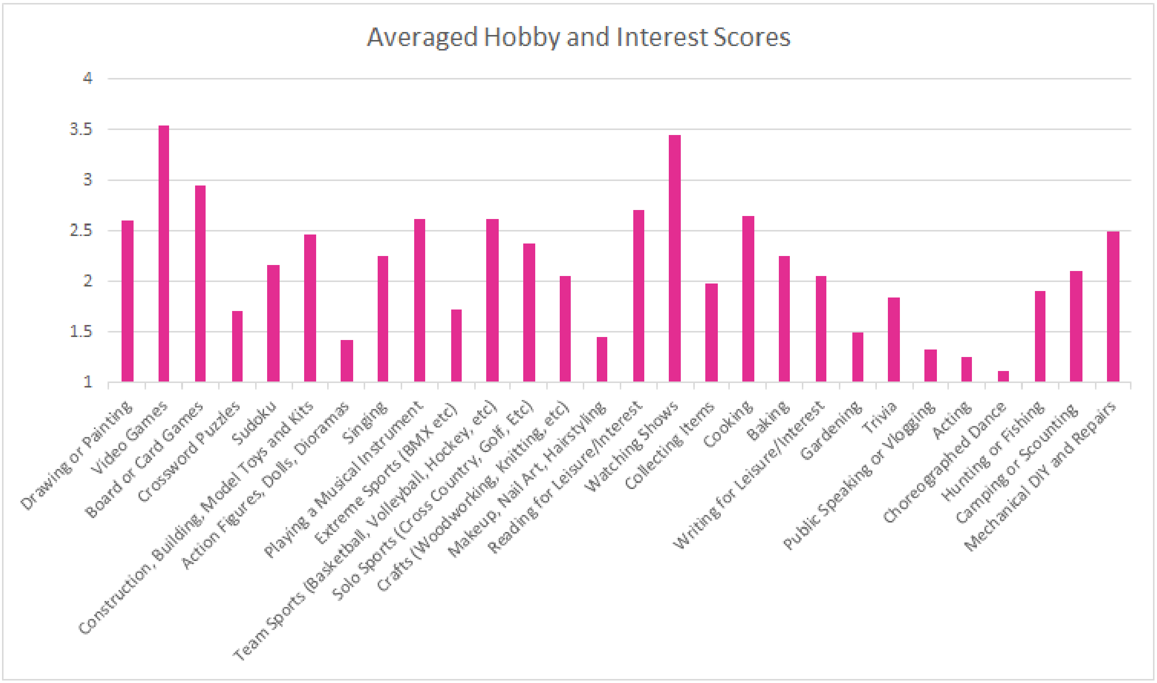


Figure 5.1: Averaged Hobby/Interest Survey Responses from CS1 Learners

Hobbies and Interests of CS1 Students

The interests of learners, and thus topical source domains that engage them, can be varied. I provided a survey worth extra credit to learners in my CS1 course regarding their hobbies and interests. This survey's questions can be found in Appendix F. 63 student responses were collected from those consenting to participate in research.

The survey presented hobby/interest categories with a 5-point Likert scale corresponding to if the learner engaged with this activity "Never" or "Most of the Time". Scores were averaged across all participants to show the general leanings of classroom interest. Graph 5.1 shows the results of this analysis.

The most common hobbies indicated by learners were Video Games (3.54), Watching Shows (3.44), and Board or Card Games (2.95). The least common hobbies were Choreographed Dance (1.13), Acting (1.25), and Public Speaking or VLogging (1.33).

Not a single score is exactly 1, which would indicate that **no** surveyed learners engaged with the activity. This means that every activity described in the survey had at least *some* engagement as a hobby or interest for some learner(s).

Learners also wrote in several additional hobbies/interests and their personal rankings of them. Several of these appeared to fit existing categories provided. Barring hobbies/interests that were *exact* re-writes of a choice provided in the survey, additional hobbies included: Yu-Gi-Oh[2], Soccer, Skiing, Traveling, Exercising, Sleeping, Tabletop Roleplaying Games, Athletics, Running, Hiking,

[2]A collectible trading card game, for readers who have not had time to duel

Sports Broadcasting, VEX[3], Rubik's Cube, Weightlifting, Guns, Anything Security Related, Chess, Photography, and UI Design.

To promote topical engagement and interest, it is important to be aware of the wide range of hobbies and interests learners enjoy. While no activity on this list was a 1, there was also no activity that was a 5 (or even a 4) across the board. No area of interest is perfectly suited to *every* learner. This range of interests punctuates the value of source domain consideration in analogy. Some analogy sources may be mundane to one learner but excite the next. Drawing from an assortment of source domains can encourage more learners to take interest in the material. The ability to pivot the source of an analogy to a specific learner's interest can help that learner establish a personal connection to the material when individual assistance is being provided.

5.2.2 Humor

Incorporating humor can inspire interest and engagement from learners, and humor can certainly be incorporated into analogy. The Instructional Humor Processing Theory [159] explores better understanding of how learners interact with humor in the classroom. Humor can have three outcomes for the learner: they do not recognize an incongruity and cannot recognize an attempt at humor; they recognize the attempt but do not "get it" due to inability to resolve the incongruity; or they are able to understand and make sense of the humor presented [159]. The third outcome is of course desired: a learner is able to understand and make sense of the humor. The resolution of incongruity in humor is also an analogical reasoning process. One must recognize *why* that that relation was drawn in the context, and how it relates.

Central processing in cognition requires learners to generate information and elaborate on an idea. This is certainly desirable engagement with classroom material. Humor can promote central processing by forcing attention to resolve the incongruity presented [159]. Central processing can even promote modification of attitudes and behaviors [159]. The form of processing humor promotes is important in the adaptation and evolution of mental models.

Analogy provides the opportunity for humor with source domain selection. So long as humorous elements are structurally relevant, there is no rule that analogy cannot make the learner laugh or smile. Humor related laughter and smiles, in fact, show active engagement and processing of the information. As analogical reasoning requires active processing to promote generalization and prediction, well-incorporated humor can provide engagement benefits for learning.

[3]A robotics competition

Chapter 6

Emblema: Internet Memes and CS Knowledge

Internet memes (hereafter, just "memes") are a social form of information transfer where an individual shares a template with contextual content added. The template holds specific meaning and structure, framing the content that individuals add. New knowledge can be transferred and experiences can be shared in this way. Countless meme templates exist, and yet many are interpreted almost intuitively, especially by digital native generations.

This research explores memes as a vehicle for learning with analogical reasoning. Programming content is incorporated to specifically investigate analogical reasoning surrounding programming concepts. This research presents evidence that memes and active discussion of them in a structured pedagogical environment may provide learning gains as well as valuable engagement and interest for learners.

6.1 Memes as Connective Structures

Popularized memes often consist of a common template to which a group has ascribed meaning. While the common template and "culture of form" is owned by the group, individuals develop a meme "expression" — often from personal experiences that they wish to share [122]. Individuals expressing themselves creatively within relational structures agreed upon by the group can foster a sense of belonging and identity within a group.

Memes also provide "snapshots" of cultural knowledge assimilation and encoding. Memes require assimilation of a knowledge body, and encoding of that knowledge in a way that derives meaning in distinct new meme images. Cultural knowledge assimilation is necessary in becoming a programmer, as one must map the culture of programming to their existing knowledge [4]. To create or understand the expression of programming memes, one must engage in this cultural knowledge mapping.

Individual memes are often used to convey their creator's emotions [114]. A meme is not meant to just state something, but often to convey some *feeling*. Memes often channel heightened emotions, which can tend toward negative representations: frustration, confusion, anger, pain, and sadness. Still, many memes also express joy, success, and humorous self-reflection. The self expression and emotional response memes create encourages their use among groups. This allows one to feel not only a sense of belonging to "the group that understands memes", but allows empathizing and emotion processing as part of that group. Emotional reactions are strong motivators and learning triggers, which highlights potential value in memes as a pedagogical tool.

Nye [127] indicates through developing their model a relationship between Bandura's observational learning process and memetics. The Bandura's model [5] suggests knowledge passes through attention, retention, motivation, and production. Nye's exploration of memes was utilizing an AI, but this model can be applied to human agents. A meme grabs one's attention (sent by friend, saw on Internet). It is viewed, and ideas are processed. The viewer retains some knowledge of the meme to draw meaning, and stores a subset of that. The viewer is then motivated to some emotion or action. They may engage with production by sharing the meme, conversing about it, or creating one themselves. The observational learning process applies to memes — but can memes be applied to learning?

Exploring the definition of an internet meme, Diaz [13] specifically draws back to the origin of the word meme in Dawkin's explorations of genetics [28]. Diaz describes characteristics that are "mutated" or "passed on", as well as constants. This sheds light on the imperative of memes as culturally held expressions that are individually created: something is allowed to change, but something else stays the same. Relationships exist within the meme image, expectations of behaviors between elements, and elements which are allowed to change to fit this expectation. The template is an abstract schema that gains meaning through the relationships portrayed by it and the elements that are modified within.

This retention and processing of this abstracted schema allows for memes to gain plausible utility as a pedagogical tool. Viewers must develop meaning based on their schema for that meme template. The process of developing a schema requires analogical encoding between examples to discern core generalizations [57]. With the analogical encoding process crafting a schema of templates, pedagogically relevant content can be presented in a meme format, allowing for analogical encoding of the specific content knowledge. Strength of understanding regarding the template's schema may allow for complex ideas to be understood, even when content knowledge is lacking. Relevant memes presented in the learning environment alongside other pedagogical methods, may be a valuable knowledge encoding tool for learners.

Research into programming students has found that sense of belonging is often related to performance, and that supporting sense of belonging can improve retention [158]. Memes allow for a sense of communal belonging, but can work to address some perceptions of ability. Students may exhibit confidence in understanding a meme template's schema. This ability to draw correct conclusions even when they feel they lack some portion of content knowledge, may bolster a student's perception of capability. When presented in an environment where the student is learning about the topic, the confirmation regarding associations presented by the meme may increase confidence. Confronting knowledge of an association but not the reason behind it may help students feel they are capable of grasping the material, better recognizing where they may need to target questions and studying.

In addition to their encoding power, memes are social, prompting a sense of belonging and engaging emotions. These aspects can foster even greater engagement and persistence in learning, and make the intrigue of memes in pedagogy even more engaging. In one of his thesis pieces, Kirk [86] notes that "the primary content that we engage with is actually content we generate in relationship with the work". Everything we understand is in relation to our experiences - the field of phenomenology is devoted to this very idea. The meme itself isn't the content to be concerned with, but rather, what the viewer takes away from the meme based in their experiences and understanding. Memes situated in pedagogy may allow for a strengthening of the educational experience, and student relationships with the ideas therein.

6.2 Study Design and Methodology

I conducted a study to observe whether the relational structure of memes could increase understanding of pedagogical content knowledge. This interview study was conducted to gain insight on how participants reasoned about the meaning of meme images. Memes with programming content were presented in this study to identify what meaning participants inferred about programming concepts outside of the classroom, given the memetic image and their understanding of its relational structure alone.

Participants were recruited via word of mouth and advertisement within courses and clubs across campus. Participants scheduled a one-hour time block to take part in a guided interview, were provided a consent form, and encouraged to ask any questions they may have. Consent to record audio for transcription was obtained.

Through the guided interview process additional questions were raised with specific participants as opportunities presented themselves. Clarifying or summarizing questions were also asked. This is in line with a dialectic form of study as seen in Lave et al [89], which allowed for gap-closing in terms of interview understanding of the participant response, as well as promoted responses due to feeling less "clinical" as a procedure. The general guiding structure of the interviews can be found in Appendix G.

All participants were presented with memes and categories in the same order. Ordering effect potential was considered for templates, but time between the same template (discussion of seven other memes) caused any ordering effect to be perceived as minimal.

Eight meme templates were used [1], presented in the following order:

1. "Is This a Pigeon?"[107]
2. "It's Free Real Estate"[105]
3. "Ight Imma Head Out"[112]
4. "Does Your Dog Bite?"[108]
5. "Who Would Win?"[106]
6. "Gru's Plan"[111]
7. "Whatcha Got There?"[109]
8. "Roll Safe"[110]

A meme of each template was presented for each of the following categories. All memes in a category were presented as a set. To give an example of memes within each of these categories, I present a

[1] As Internet meme images are an area with tenuous copyright precedence and ethical considerations, I have taken my lead on publication considerations from Milner [114]. In his book on memes in popular culture he includes his own process and concerns regarding publication, which I have used to inform my own usage here. The origin of each meme's imagery (episode, artist, show) was sourced and appropriately cited. All meme images sourced were from produced entertainment to mitigate consideration of an individuals' likenesses being captured without consent. Representations of the these images included in this research have significant modification for educational purposes and to ensure the spirit of fair use. This includes desaturation of the original images and inclusion of coloration, labels, and indicators to facilitate the research arguments.

memetic style image that I developed for illustrative purposes in this section, the "Curious Squirrel" meme.[2]

1. **Blank** : A template, no text or content added. Figure 6.1 shows an example.
2. **Non-Programming** : The meme is filled in with content that is not related to programming. Figure 6.2 shows an example.
3. **Programming** : The meme is filled in with content that relates to programming in some way. Figure 6.3 shows an example.

Category ordering was distinctly chosen. Blanks were shown first to recognize any existing conceptions the participant had about the specific template. Non-programming versions were then shown to explore the subject's reasoning process to obtain information about the meme. Finally, programming memes were shown to see the subject's application of that reasoning process to the domain of programming, in order to understand how that reasoning might impact their understanding.

Participants were also asked if they were willing to review four additional memes, categorized as **"Remix"**. These intentionally modify the format in some way. All were non-programming related. These allowed additional insight surrounding how modifications changed ability to encode or understand the meme.

The remixed memes explored the following modifications:

1. **"Memeception"** : A meme is used within another meme. "Is This a Pigeon?" was combined with "It's Free Real Estate", with additional non-meme images being added to the context.
2. **Modified Background** : "Ight Imma Head Out" had the house background replaced with a background that related to the meme's content.
3. **Analogous Meme** : A new template was presented: "Plankton's Plan". This followed the format of "Gru's Plan".
4. **Image Paste** : "Roll Safe" was presented in a context that had the man's face replaced with a different face.

This study was originally designed for on-campus interviews to be conducted with physical consent forms and signatures. However, as the COVID-19 pandemic caused campus closures, the study was moved to remote via Google Hangouts calls. Participants were sent a calendar invite which included a PDF version of the consent form to review, and a link to a Google Forms version of the consent form to provide their digital signature. No other aspect of the study required a change in transitioning to remote.

[2]This is not a "real meme" in the sense that it would be found online or used by a broad group of people, nor was it used in the study. It is used here exclusively to represent the categories of memes presented for publication purposes. I took the photograph "Curious Squirrel" uses and therefore have rights to it, per the previous footnote regarding publication use of popular culture memes.

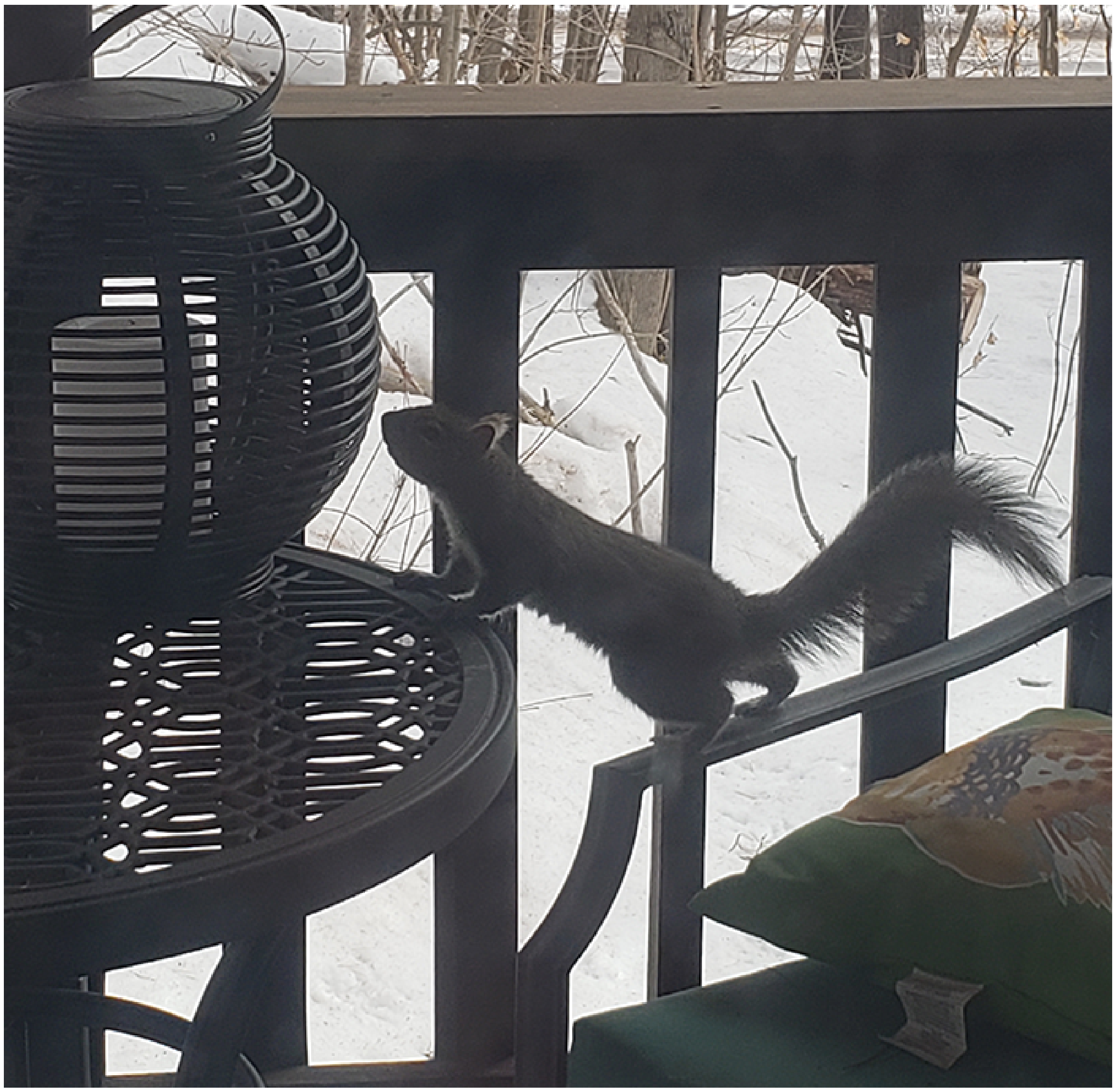


Figure 6.1: Blank: “Curious Squirrel” Example

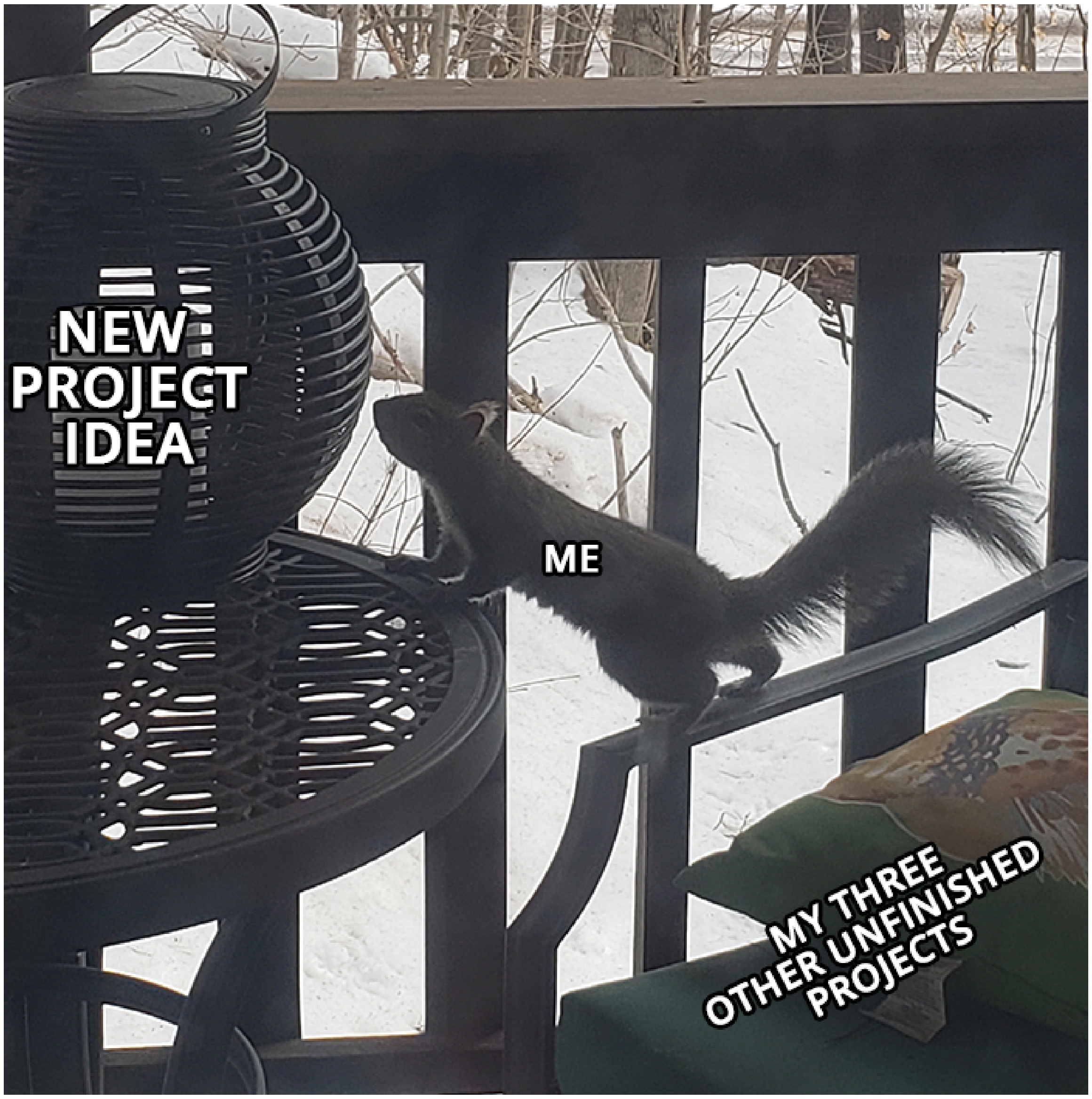


Figure 6.2: Non-Programming: “Curious Squirrel” Example

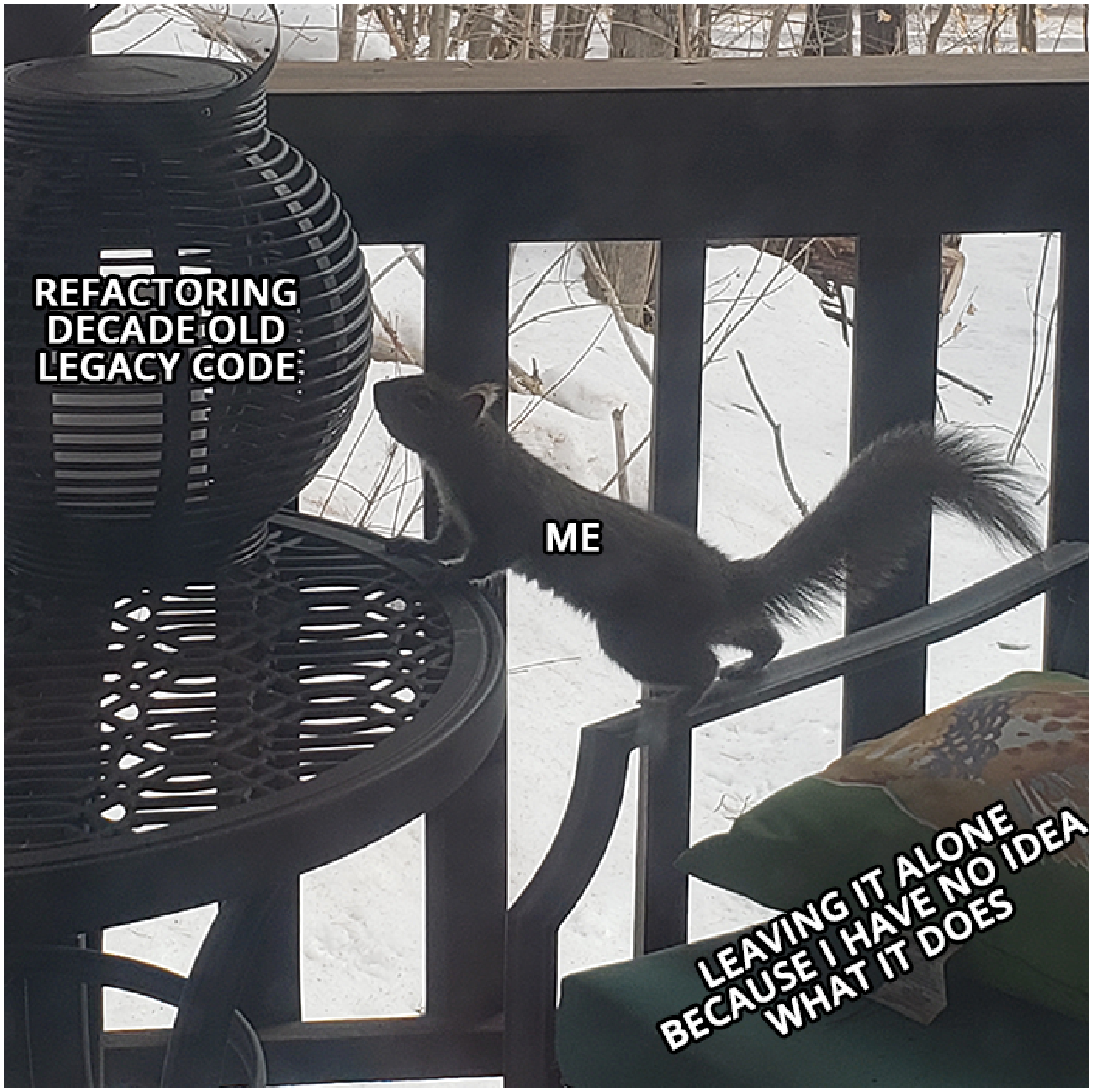

REFACTORING DECADE OLD LEGACY CODE
ME
LEAVING IT ALONE BECAUSE I HAVE NO IDEA WHAT IT DOES

Figure 6.3: Programming: “Curious Squirrel” Example

6.2.1 Data Collection

One pilot participant was run, followed by thirty actual participants. Eleven interviews were conducted in-person on campus and the remainder online. Of the thirty participants, the first five in-person participants did not have notes taken in a similar manner to the other twenty-five. For review here, these five participants were also considered pilot participants.

We can observe pre-survey metrics for all thirty participants, however, which indicated the following.[3]

Gender Identity	**Male** 16	**Female** 14
Programmed Prior	**Yes** 26	**No** 4
Familiarity With Memes	**Mean** 4.383	**Mode** 5
Understanding of Memes	**Mean** 4.167	**Mode** 5
Programming Experience	**Mean** 3.85	**Mode** 5
Age	**Mean** 20.387	**Mode** 18

While not all participants indicated a major, those that did were primarily concentrated in Computer Science and Software Engineering. Computer Engineering, Mechanical Engineering, Computer and Electrical Engineering, and Biomedical each had more than one participant. Single participants identified as Environmental Engineering, Business Analytics, Economics and Finance, and Cybersecurity.

Participants who disclosed year of school primarily identified as first-year, with fourth-year being second most common. A handful of students identified as third-year, but none in the data set identified as a second-year student.

6.2.2 Analysis

Throughout the interview, timestamps were noted by the interviewer for later review. This approach was chosen based on Brown's suggestion in the design of observational research [10]. Brown noted that having investigators and/or participants note events of interest allowed "selection before selection", reducing bias by notating interesting phenomena prior to any investigation of their meaning. All interviews were approximately an hour, and transcription was pinpointed to these timestamps in alignment with Brown's suggestion in order to reduce the initial analysis space.

In transcription of participant timestamps, some were collapsed together due to ongoing dialogue that relevantly connected them. Of the 25 timestamped participants, 1,169 entries in total were transcribed. Due to the size and richness of this data set even with the reduction provided with Brown's suggestion, a full grounded theory analysis is beyond the scope of this work. Instead, this work presents a preliminary analysis of emergent themes from the transcription process. Transcription of the timestamped data caused preliminary themes to emerge.

6.3 Preliminary Observations

Based in the preliminary analysis, three major themes of interest to using memes as analogical reasoning tools were identified.

[3]Responses are on a 5-Point Likert Scale with 1 being "Least" and 5 being "Most".

Full quotations mentioned in this analysis to exemplify the observations can be found in Appendix H.

6.3.1 The Relational Structure of Memes

The internalized relational schemas participants had regarding the meme templates was overwhelmingly observable. In asking participants to reason about their conclusions, many indicated specific structural rules. These rules exist within the template as a purely relational system. *Any* information placed in the meme becomes a target domain, with the abstract structural knowledge being used as the source.

Knowledge of memes having *some* relational structure was a driving motivator for this study. The specificity of rules and relationships governing meme use, and how participants were able to describe them, was much stronger than initially considered.

Using participant observations, a breakdown of the relational schema for each of the memes follows here. This relational structure can be further described using structure mapping grammar [49], which is indicated as well. An image visualizing the structure mapping grammar is presented. The content of the programming memes is also presented exclusively in the grammar form to highlight this relational capacity.

To exemplify how information in this section is laid out, I present the "Curious Squirrel" example using the same approach that will be provided for the study memes, which is shown in Figure 6.4.

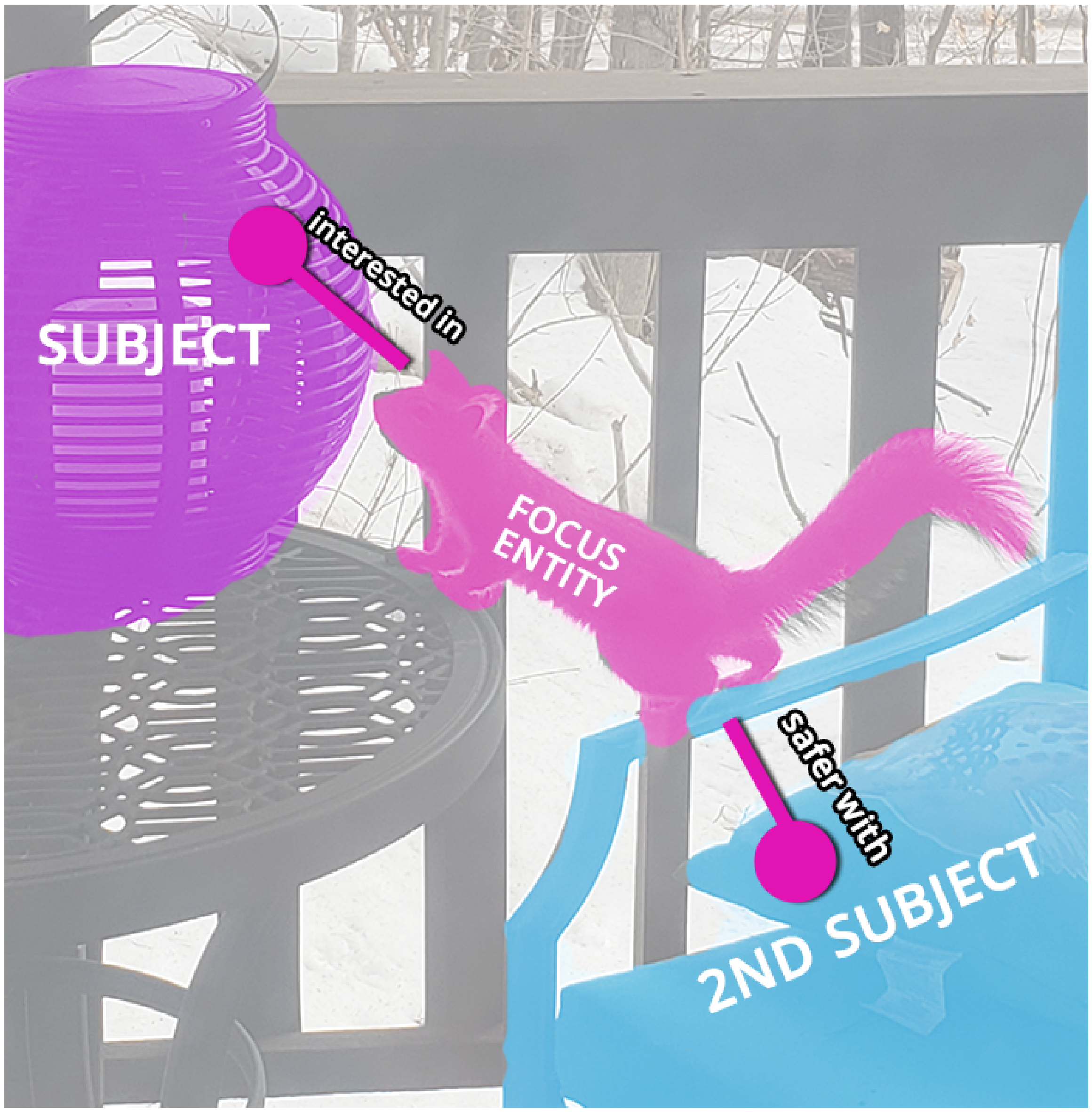


Figure 6.4: Breakdown: Relational Structure Example of “Curious Squirrel”

“Curious Squirrel” Relational Grammar

1. INTERESTED-IN (focus-entity, subject)
2. SAFER-WITH (focus-entity, 2nd-subject)

“Curious Squirrel” Programming Meme as Structure Mapping Grammar

1. INTERESTED-IN (me, “refactoring decade old legacy code”)
2. SAFER-WITH (me, “leaving it alone because I have no idea what it does”)

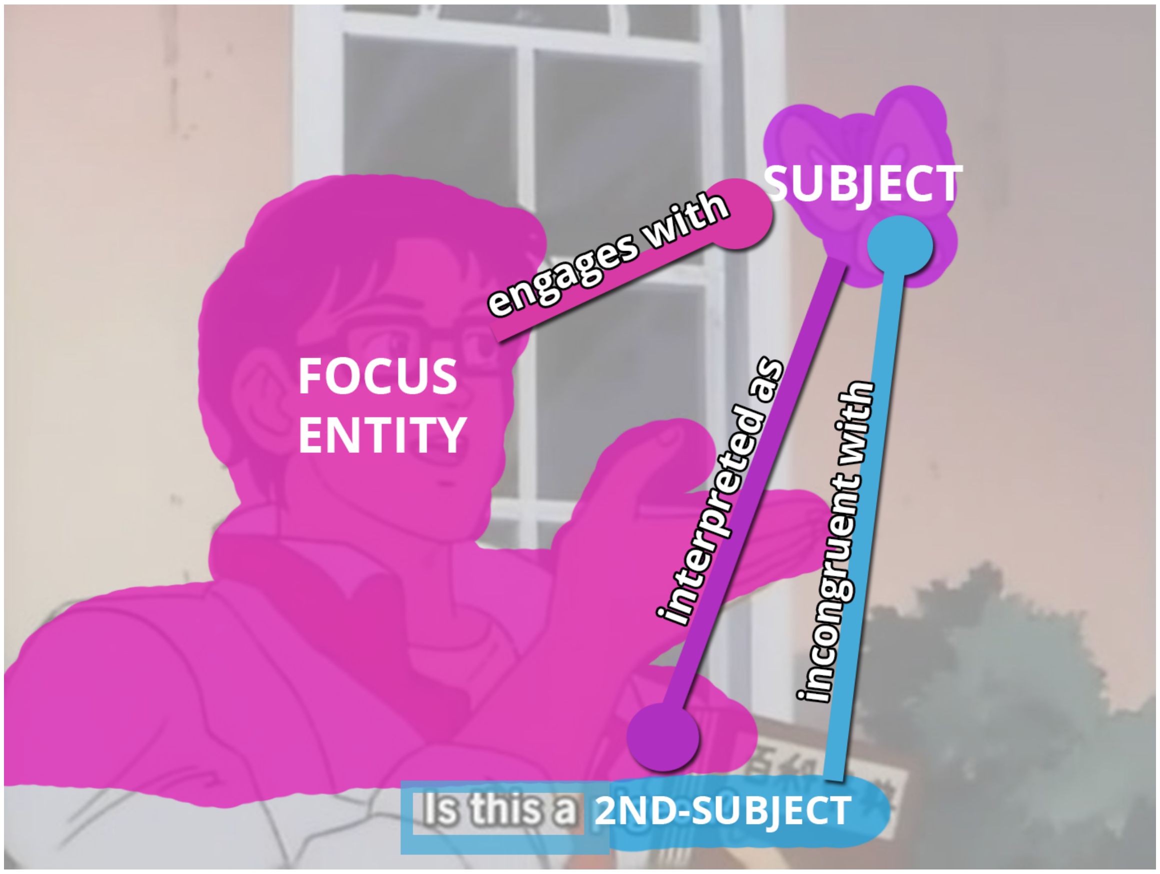


Figure 6.5: Breakdown: Relational Structure of “Is This a Pigeon?”

“Is This a Pigeon?” Relational Grammar

1. ENGAGES-WITH (focus-entity, subject)
2. INTERPRETED-AS (subject, 2nd-subject)
3. INCONGRUENT-WITH (2nd-subject, subject)
4. INDICATES (focus-entity, AND(“Is this a”, 2nd-subject))

“Is This a Pigeon?” Programming Meme Grammar

1. ENGAGES-WITH (programmer, if-statements)
2. INTERPRETED-AS (if-statements, AI)
3. INCONGRUENT-WITH (AI, if-statements)
4. INDICATES (programmer, AND(“Is this a”, AI))

Evidence of "Is This a Pigeon?" Structure

This meme template provided some very interesting observations with regard to the relational structure. What's termed as "incongruence" is not completely incompatible as the term might suggest. Participants observe more so that there is *some* aspect that is incompatible between the two items, and that the meme structure is meant to foster that comparison. The comparison also requires them to consider what is similar, because the incongruity is not a complete difference between the two. Participant 7 describes their lack of understanding the distinction, but as they learned more about artificial intelligence and continued to see these formats, came to better understand the humor that the incongruities are meant to convey.

> P7: [...]the way I see it is like a venn diagram between AI and if statement, and there is some overlap in that diagram in terms of like, artificial intelligence using if statements [...]

Participant 10 further solidifies this conception. They note that the point is that the items are not completely disparate, so the statement isn't "wholly" inaccurate: something ties the two ideas together. However, it certainly isn't correct due to the incongruities. Without using the term conditional as participant 7 did, they describe the concept of conditionals. They also note another interesting theme — the incongruence is usually due to the 2nd subject being somehow more complex to consider than the first. The humor comes from the perception that the "simpler" idea is the same as the more "complex", or as the participant states, "high level" idea.

As memes allow sharing of experiences and emotions, participants such as participant 12 also generated scenarios for how this meme may "play out" in real life, indicating how a real person might arrive at this incongruence.

Participant 27 encapsulates the humor in the incongruity between subject and 2nd-subject quite well:

> P27: Yeah or they just think it's funny because there's like a small similarity that they share that they try and say that they're the same as a joke.

There was also evidence that the relational structure of the meme is so powerfully understood, that domain knowledge is not necessary to recognize the premise. Participants 7, 12, and 29 specifically discuss non-programmers considering the programming variant of the meme. They reasoned that they would be able to understand the premise due to the meme's structure, even if they were not able to make the corollary to the key ideas behind the relationships, such as conditional logic.

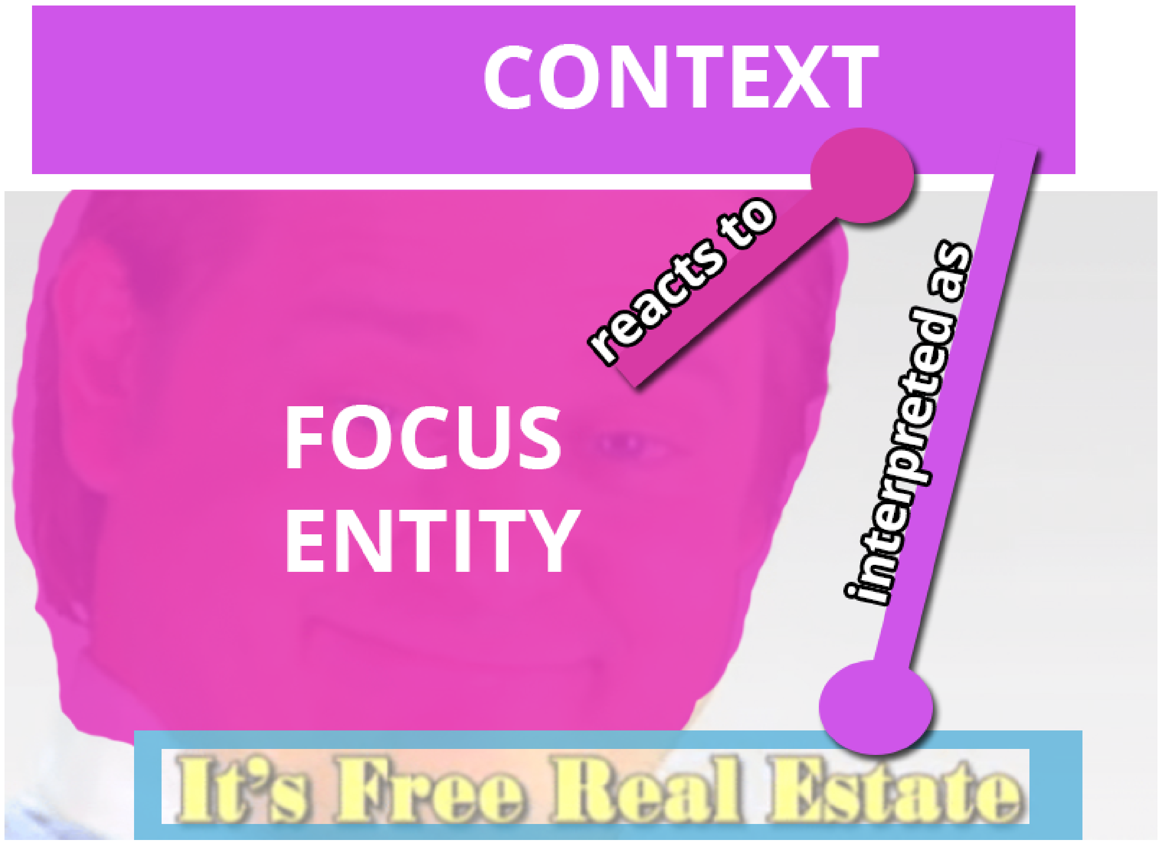


Figure 6.6: Breakdown: Relational Structure of "It's Free Real Estate"

"It's Free Real Estate" Relational Grammar

1. REACTS-TO (focus-entity, context)
2. INTERPRETED-AS (context, "It's Free Real Estate")
3. INDICATES (focus-entity, "It's Free Real Estate")

"It's Free Real Estate" Programming Meme Grammar

1. REACTS-TO (JVM, "RAM: *exists*")
2. INTERPRETED-AS ("RAM: *exists*", "It's Free Real Estate")
3. INDICATES (JVM, "It's Free Real Estate")

Evidence of "It's Free Real Estate" Structure

One key indication of the structure of this meme by participants was an element lacking in the blank meme presentation: context. Several participants refer to the context information prior in explaining the programming case of the meme. Participant 17 indicates when seeing the template that there should be information prior.

> P17: I feel like there needs to be like another meme before it that like says what he's talking about

Exemplifying the relational structure of memes, participants often used other domains in working to explain the relations they were seeing. Participants 7 and 24 both reference the non-programming meme, which involves a cat indicating a box set on the floor is free real estate. Participant 7 provides further evidence of the structure, indicating the setup has implied RAM as an element *present* in the scenario, but not the focus entity.

Many participants also indicated use of added cues to guide them in understanding elements within the relational structure. "JVM" was not an understood acronym by many participants, but the eyes of the focus entity had been "corrupted" as participant 9 describes to have the Java logo over them. Participants noted using the eyes to understand the focus entity (JVM) must be Java, at least to some degree. Participant 20 recognizes the term Java Virtual Machine, but only after being asked where they retrieved the term Java from, and mentioning the eyes.

> P9: So the it's free real estate they corrupt the guy's eyes with Java so um so well I don't necessarily unders- I don't know what JVM is [...]

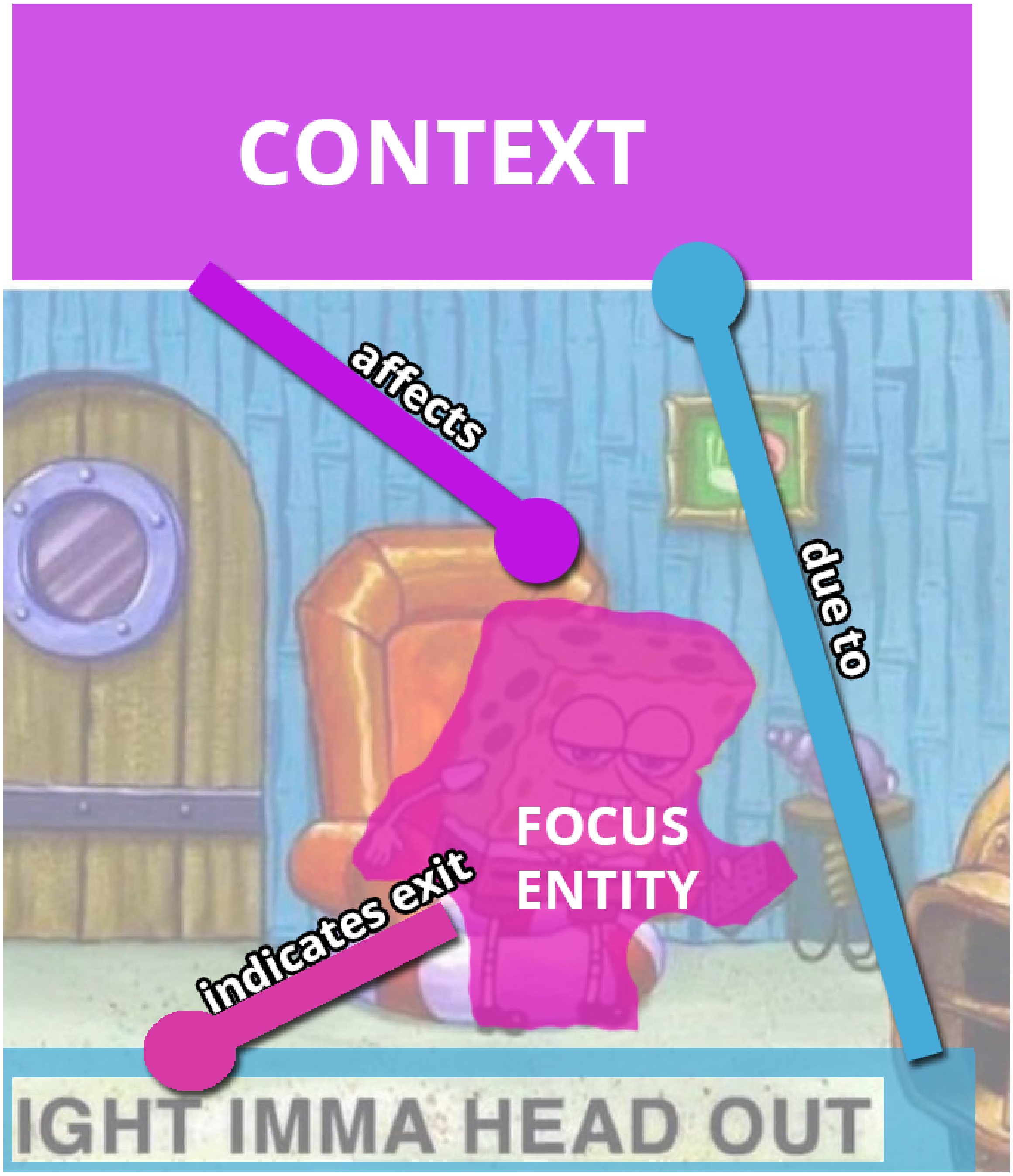


Figure 6.7: Breakdown: Relational Structure of “Ight Imma Head Out”
“Ight Imma Head Out” Relational Grammar

1. AFFECTS (context, focus-entity)

2. INDICATES-EXIT (focus-entity, “Ight Imma Head Out”)
3. DUE-TO (“Ight Imma Head Out”, context)

“Ight Imma Head Out” Programming Meme Grammar

1. AFFECTS(“Local Variables When They See A “}” :”,
   local-variables (IMPLIED))
2. INDICATES-EXIT (local-variables (IMPLIED), “Ight Imma Head Out”)
3. DUE-TO (“Ight Imma Head Out”,
   “Local Variables When They See A “}” :”)

Evidence of “Ight Imma Head Out” Structure

Participants 12, 15, and 21 make specific reference to the wording in this meme, and how it indicates the focus entity’s action is based on the context. Some participants indicated, like participant 15, that the action wasn’t *physical* as the meme might suggest, but again a relation. Leaving is a key idea due to the “head out”, but this action does not have to be physical, only relevant in some way to the context.

Even participants who were less certain about the intended programmatic meaning were able to use these cues to reason about the relationship. Participant 17 shows an understanding of the domain concept related to the meme, but expresses uncertainty if they have recalled correctly. Participant 22 clearly is unaware of the programming implications, but tries to reason based on the relational structure. They specifically note “not for you” in regard to the variables and their context, showing a symbolic understanding of one making an exit due to feeling unwelcome or at the end of their area of comfort.

> P22: [...] I don’t know why it makes the local variables not...work it looks like? Whatever kind of variable that would be it’s not for you

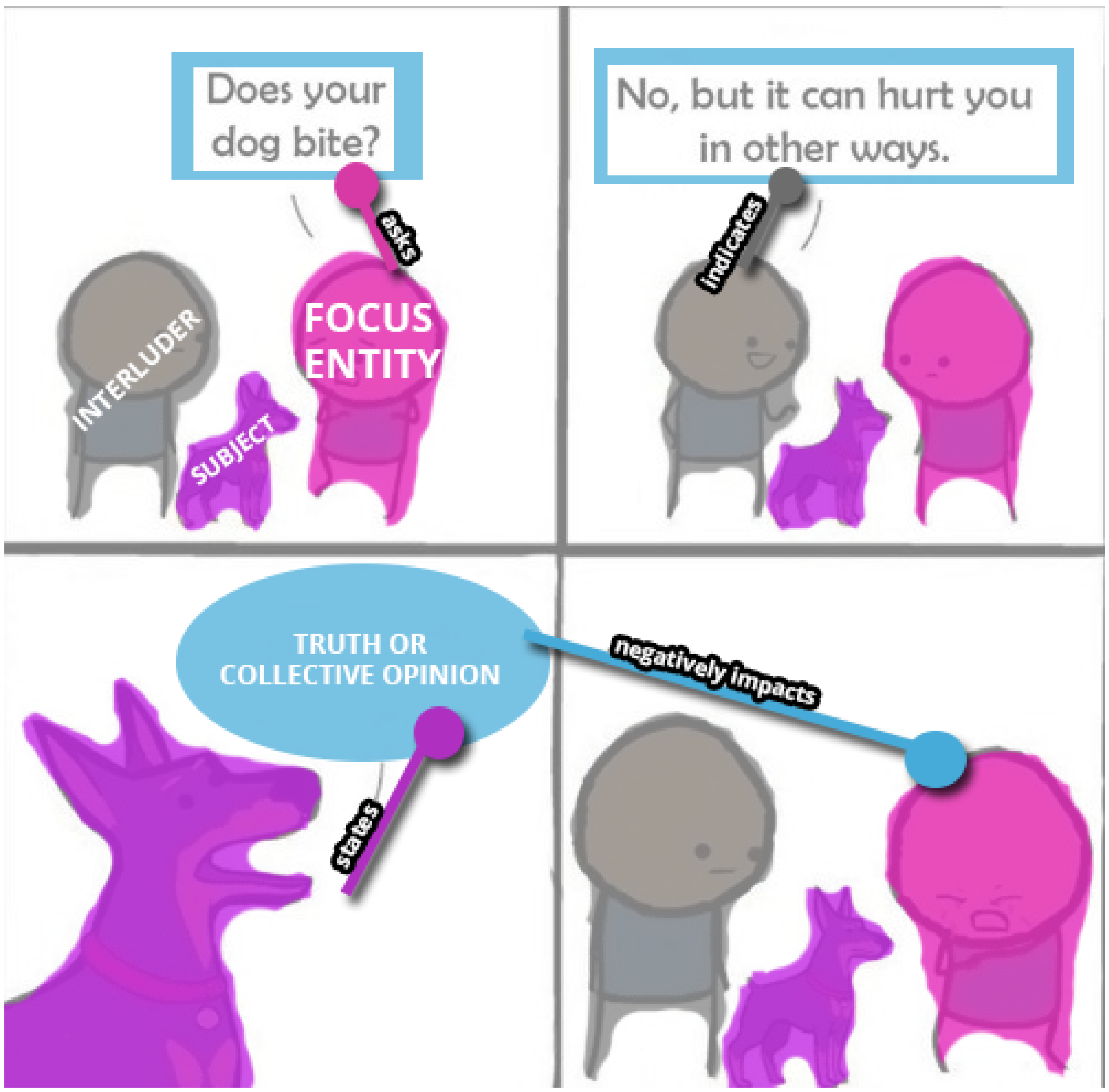


Figure 6.8: Breakdown: Relational Structure of 'Does Your Dog Bite?"

"Does Your Dog Bite?" Relational Grammar

1. ASKS(focus-entity, "Does Your Dog Bite?")
2. INDICATES(interluder, "No, but it can hurt you in other ways")
3. STATES(subject, truth-or-collective-opinion)
4. NEGATIVELY-IMPACTS (truth-or-collective-opinion, focus-entity)

"Does Your Dog Bite?" Programming Meme Grammar

1. ASKS(reader (IMPLIED), "Does Your Dog Bite?")
2. INDICATES(meme-creator (IMPLIED), "No, but it can hurt you in other ways")
3. STATES(MATLAB (IMPLIED), "Array indexes start at 1")
4. NEGATIVELY-IMPACTS ("Array indexes start at 1", reader (IMPLIED))

Evidence for "Does Your Dog Bite?" Structure

Despite the programming meme never making mention of the number zero, several participants offer up the value of zero as part of their reasoning process. The dog clearly indicates "arrays start at one". This suggestion of zero shows their understanding that clearly, the focus entity believes something different than what the dog is saying. Participant 6 laughs as they realize they may have at first believed the dog. Participant 8 indicates a lack of knowledge, but still states zero specifically as part of their reasoning process. They also indicate that the dog may have some truth to their statement, which echos participant 9, who also states uncertainty on the topic but says they know one and zero is a debate within computing. This indicates subtlety in that the dog is not saying *something completely wrong*, but something that may be difficult to hear or only true within certain contexts.

Other participants with programming knowledge indicated they felt the dog was wrong, and that's why it was painful to hear what it had to say. This may seem like an incorrect relational structure. However, this actually underscores the structure of the meme — the dog's statement must be true in some way *or believed true to some group.* The nature of that truth negatively impacts the focus entity. These participants illustrate a direct relationship with the focus entity, who is hurt by the dog stating arrays start at one. They share the focus entity's frustration at this statement, belonging to the group who believes this is not true. These participants are being *exemplified* in the meme, highlighting its ability to convey experiences and foster sense of belonging. Participants 14 and 30 offer the focus entity's perspective directly. Participant 18 gives insight to the subtlety that what the dog is saying is true to *someone*, suggesting that another language may have this as a feature, giving truth to the sentiment but highlighting its frustrating nature.

Participant 8 directly describes the focus entity, and how modifications can be made to their appearance in order to provide additional context. This shows evidence for a function ascribed to many of the memes: a focus entity that is implicitly assumed to be the reader exists, but evidence within the meme's content can modify this interpretation.

> P8: He's the subject of this whole meme here, um and a modification of him modifies who the meme is really tailored for [...]

Participant 23 adds an interesting perspective — suggesting the dog *is* representing a language which starts at one, MatLab. This allows further perspective as to the nature of a collective opinion or truth. If MatLab is "stating" this information, it is situated as a truth of MatLab, highlighting the relations we've seen prior. Several participants mentioned MatLab directly in discussing the meme, despite MatLab not being present in the context. Only participant 23 specifically suggested that the subject *be* some entity or group which poses this as truth.

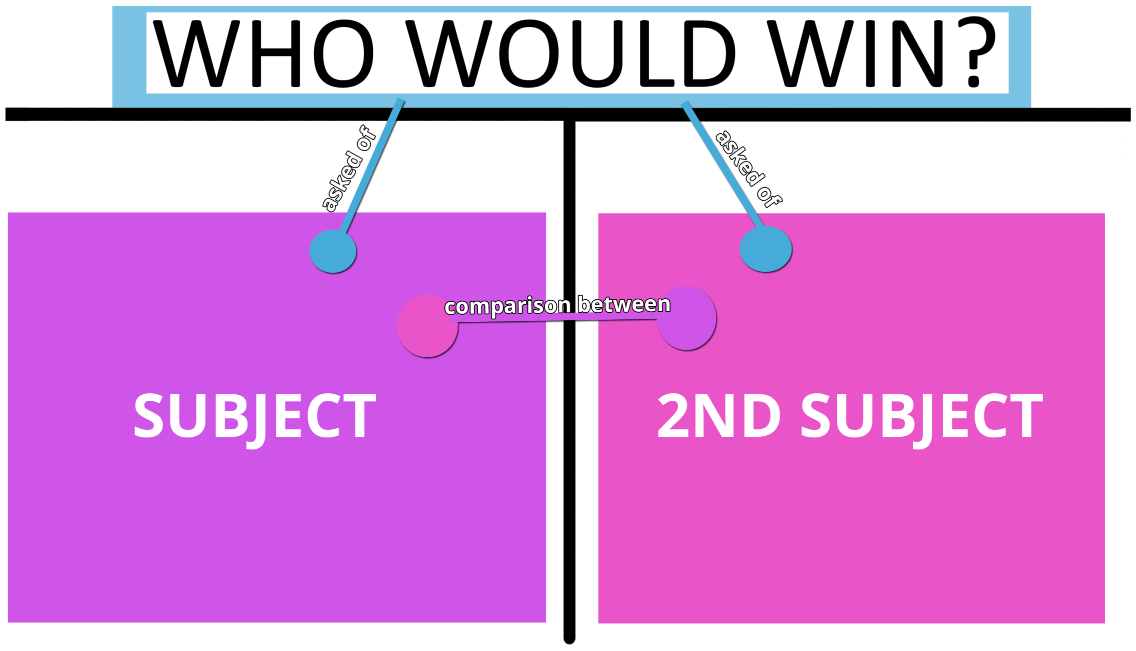


Figure 6.9: Breakdown: Basic Relational Structure of “Who Would Win?”

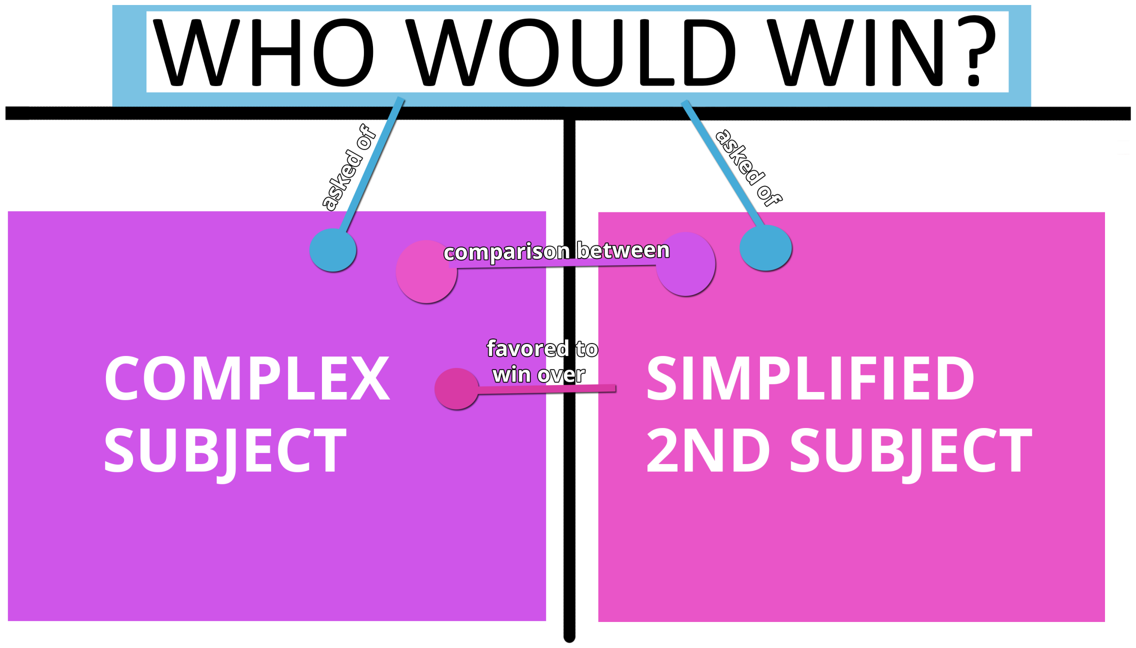


Figure 6.10: Breakdown: Alternative Relational Structure of “Who Would Win?”

“Who Would Win?” Alternative Relational Grammar

1. ASKED OF(“Who Would Win?”, AND(complex-subject, simplified-2nd-subject))
2. COMPARISON-BETWEEN(complex-subject, simplified-2nd-subject)
3. FAVORED-TO-WIN-OVER(simplified-2nd-subject, complex-subject)

“Who Would Win?” Programming Meme Grammar

1. ASKED OF(“Who Would Win?”,
   AND(“A computer program with millions of lines of code”, “One curly boi with no friend”))
2. COMPARISON-BETWEEN(“A computer program with millions of lines of code”,
   “One curly boi with no friend”)
3. FAVORED-TO-WIN-OVER(“One c u r l y boi with no friend”,
   “A computer program with millions of lines of code”)

Evidence for “Who Would Win?” Structure

Both the non-programming and programming meme presented to participants used the more specialized Alternative Relation. Several participants noted the general structure in describing the blank template, while others did observe the alternative structure. Both the general and alternative structure share a comparison relationship, but the alternative structure adds specific additional rules indicating the favored victor. Participants 21 and 29 both indicate the idea of complexity or formality being on the left, with something funny and smaller on the right. In the alternative structure of the meme, the right is favored to win over the left, which both participants indicate use of in their reasoning. Participant 9 also notes that the way the text is formatted helps indicate this structure — that the simplified structure uses slang which makes it humorous, and thus, this promotes its favorable odds.

Many participants also specifically referenced the “no friend” text as an indication of the situation that caused the right side to win. They noted that “no friend” helped convey that the closing curly bracket had been forgotten from the opening curly bracket pictured, using that to reason about why that side was favored to win. Participant 8 indicates an uncertainty as to what happens without a closing curly bracket, but acknowledges the favorable odds for the curly bracket side through this reasoning. Participant 15 indicates their entire reasoning process, and then specifically notes the role “no friend” plays as they finish their thoughts.

Participant 15 also brings in personal experience with this problem, and highlights the competition nature. Their statement also indicates the deeper underlying structure between both meme grammars, that the items are being “compared”, thus “competing” in some way.

> P15: [...]the relationship I implied is that that bracket is in the line of code and that’s so the bracket is gonna win in this situation because it won’t compile and that’s kind of the humor.

Participants 17 and 23 provide indication of reasoning that aligns more with the general structure. Participant 17 notes that so much code is “just as bad” as an error — suggesting that the competition is more evenly matched in terms of failings. Participant 23 elaborates further on this line of reasoning, suggesting that neither of these are really “winning” conditions. They also see the competitors as separate entities as noted by “there’s nothing that says that the computer program [...] it can compile”. This indicates viewing the curly bracket as outside or separate from the program, where other participants recognized that to be a foil to the program, it must be contained in it. However, this participant also later indicated that regardless, if the curly bracket is inside the code, neither can “win” anyway, as the program fails and the curly bracket is the cause of the failure. Despite not being able to observe the humor, the participant’s knowledge of programming did in fact lead them to draw the correct inferences.

> P23: [...]So I’m sort of comparing like computing power, but a curly bracket has no computing power in and of itself, and the other comparison that I’m looking at is uh, the idea of which one is more likely to crash, but that doesn’t make sense with ”who would win” - ”who would lose” seems better [...]

An interesting note regarding memes as relational structures was made regarding this meme in the post-interview. Participant 29 specifically references the curly bracket meme in describing the idea that memes are not considered based on the content, but relational structure. Participants, even if they may refer to other memes using the same format in their explanation, do not immediately think of past meme content to formulate their reasoning. The meme as a structure is observed as isolated from the context it contains.

Further still in evidence of these general structures was forward generation of content. Given a blank template, or in some discussions of a content-filled one, participants frequently generated content to fit the meme spontaneously in order to describe it. Participant 24 spontaneously generated the below context in viewing the blank “Who Would Win?” meme, which was quite similar to the programming meme shown.

> P24: Like on the left panel would be a computer and then on the other side would be like a picture of like a clipart caterpillar and it would say like ”who would win - a computer or one buggy boy” or something

Several participants showcased this behavior across other templates as well. Many drew on a domain that was culturally relevant at the time of this study being conducted — the COVID-19 pandemic. This observation also provides further evidence to Participant 29’s claim above that previous content is not referenced. The spontaneous generation of current topics showcases participants are not just drawing from a single example, but have an abstracted relational structure in mind for how each template works.

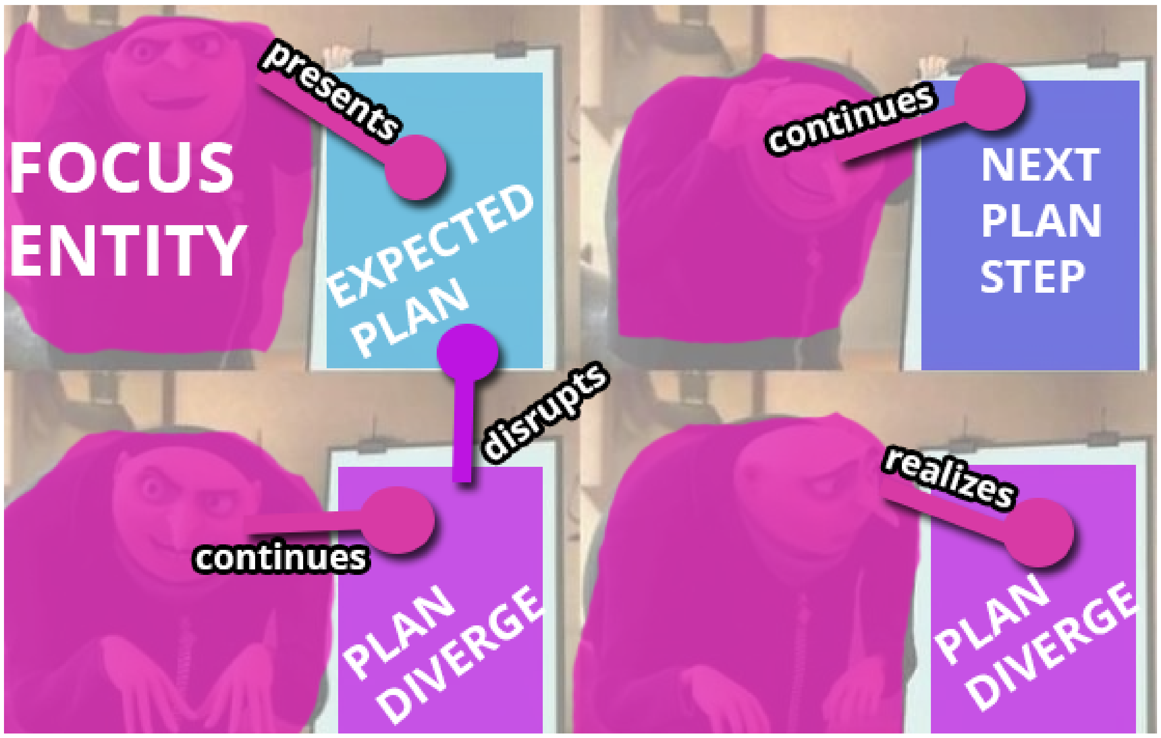


Figure 6.11: Breakdown: Relational Structure of "Gru's Plan"

"Gru's Plan" Relational Grammar

1. PRESENTS(focus-entity, expected-plan)
2. CONTINUES (focus-entity, next-plan-step)
3. CONTINUES (focus-entity, plan-diverge)
4. DISRUPTS (plan-diverge, expected-plan)
5. REALIZES (focus-entity, plan-diverge)

"Gru's Plan" Programming Meme Grammar

1. PRESENTS(reader (IMPLIED), "We Find the Bug")
2. CONTINUES (reader (IMPLIED), "We Fix the Bug")
3. CONTINUES (reader (IMPLIED), "Now We Have Two Bugs")
4. DISRUPTS ("Now We Have Two Bugs", 'We Find the Bug")
5. REALIZES (reader (IMPLIED), "Now We Have Three Bugs")

Evidence for “Gru’s Plan” Structure

This meme presented many cases of interest to one of the other observed themes. Comments and observations associated with that theme have been moved to the “Subversion and the Career of Metaphor” section as a result.

Many participant responses provided a justification for how the number of bugs changes from two to three without the focus entity noticing. These revolved around the sense that debugging was a never ending experience, often grounded in their own experiences. Participants 16 and 29 both indicated situations this might occur. Participant 22, while not describing a scenario this may occur in, directly shows the process of self insertion through the use of “you”, showcasing the implicit identification of focus entity as oneself.

> P16: [...] then it’s like oh no they have three bugs and they’re like ”oh my God when is it ending” so it *laughs* this particular meme just expresses the frustration of programmers [...]

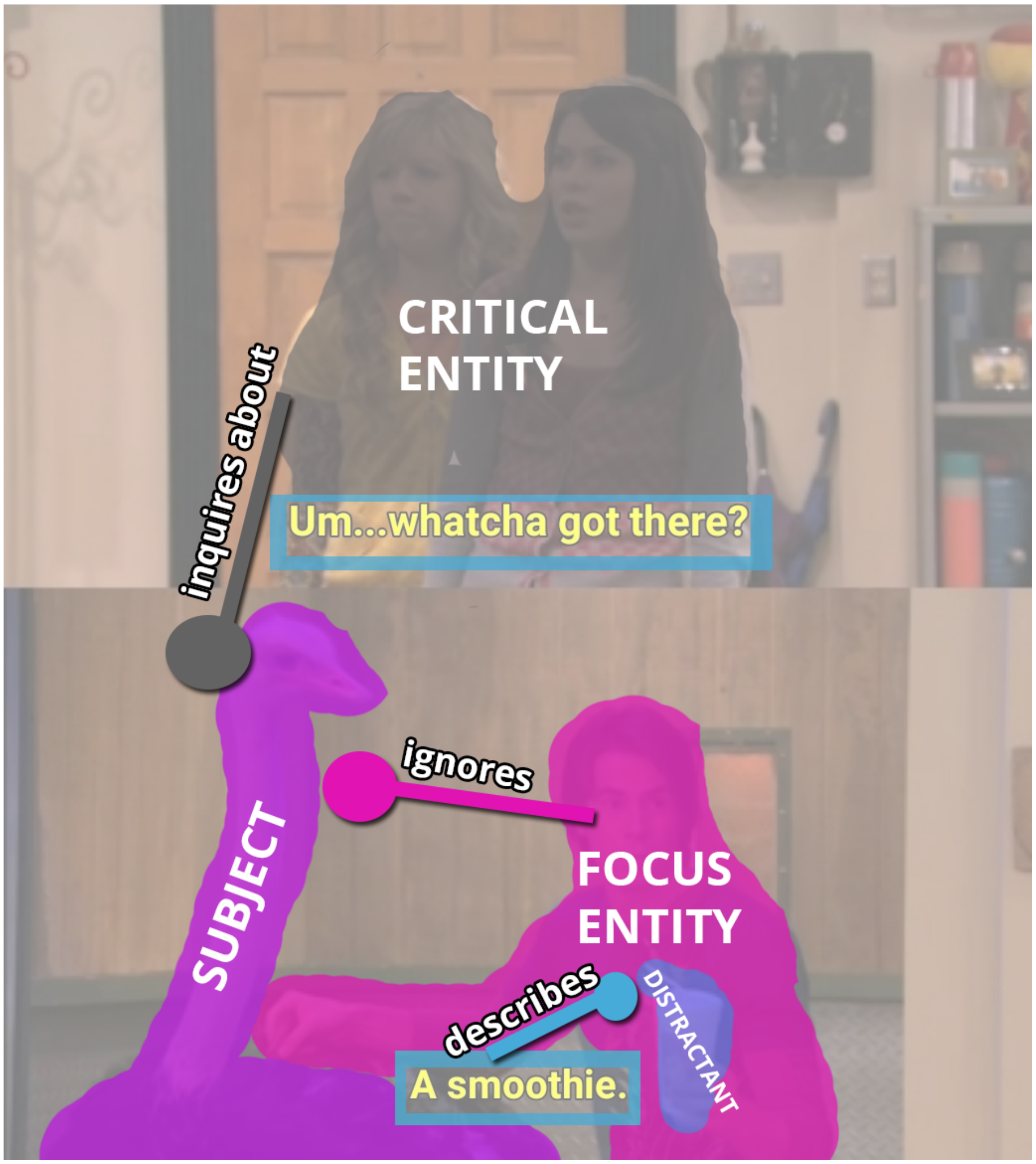


Figure 6.12: Breakdown: Relational Structure of "Whatcha Got There?"

"Whatcha Got There?" Relational Grammar

1. INQUIRES-ABOUT(critical-entity, subject)
2. INDICATES(critical-entity, "Um...whatcha got there?")

3. IGNORES(focus-entity, subject)
4. INDICATES(focus-entity, “A smoothie.”)
5. DESCRIBES(“A smoothie.”, distractant)

“Whatcha Got There?” Programming Meme Grammar

1. INQUIRES-ABOUT(programming-professors, global-variables)
2. INDICATES(programming-professors, “Um...whatcha got there?”)
3. IGNORES(me (reader - IMPLIED), global-variables)
4. INDICATES(me (reader - IMPLIED), “A smoothie.”)
5. DESCRIBES(“A smoothie.”, other-code-aspect (IMPLIED))

Evidence for “Whatcha Got There?” Structure

Many participants indicated strength of the meme structure by indicating a lack of knowledge about global variables. Participant 9 indicates directly their use of the meme format to aid in this understanding. Participant 11 suggests a definition for global variables and uses this to attribute meaning to the meme, despite their use of “probably” showcasing they are not in fact certain on that being the definition. Participant 12 takes a guess, indicating a lack of acceptance of them. Participants 14, 17 and 22 note that these “seem” like something bad or that shouldn’t be used — showing knowledge of the critical entity of the programming professor’s scrutiny toward them. Participant 16 states they haven’t used them, but attempts to reason about what they are and why they may be a problem that someone would wish to deflect from. Participant 25 goes so far as to state it is “obvious” that the global variables are being hidden based on the relational structure, but cannot ascertain why that may be.

> P17: Is there a reason you’re not supposed to have global variables? [...]

Even participant 8, who feels very uncertain, indicates that global variables are “just bad” despite no content knowledge. Their uncertainty appears to come from an inability to reason about *why*, but they recognize what they are reasoning about.

> P8: [...] Eh, global variables are just bad I guess and you shouldn’t use them but a student did, maybe for like a shortcut [...]

Participant 21 makes use of the non-programming version of the meme presented, which designated the critical entity as cops and the subject as an illegal racecar. They use this to indicate that the structure includes a sense of the critical entity believing the subject is “bad” in some way, exemplifying the entity’s “critical” nature.

Participants 7 and 19 indicate that this is “common knowledge” that one shouldn’t use global variables, with participant 7 noting their own experience with the critical entity scrutinizing them.

Some participants also indicated that even though the text consistently says “A smoothie”, one need not consider it as an “actual smoothie”. Participant 20 shows an example of the smoothie as a distracting entity, where while the text says “smoothie”, the real smoothie is a distraction a student may use, such as showing other portions of their code.

P20: [...] I don't think it would be an actual smoothie. [...] I guess they'd be like showing other parts of the code or like ignoring that fact [..]

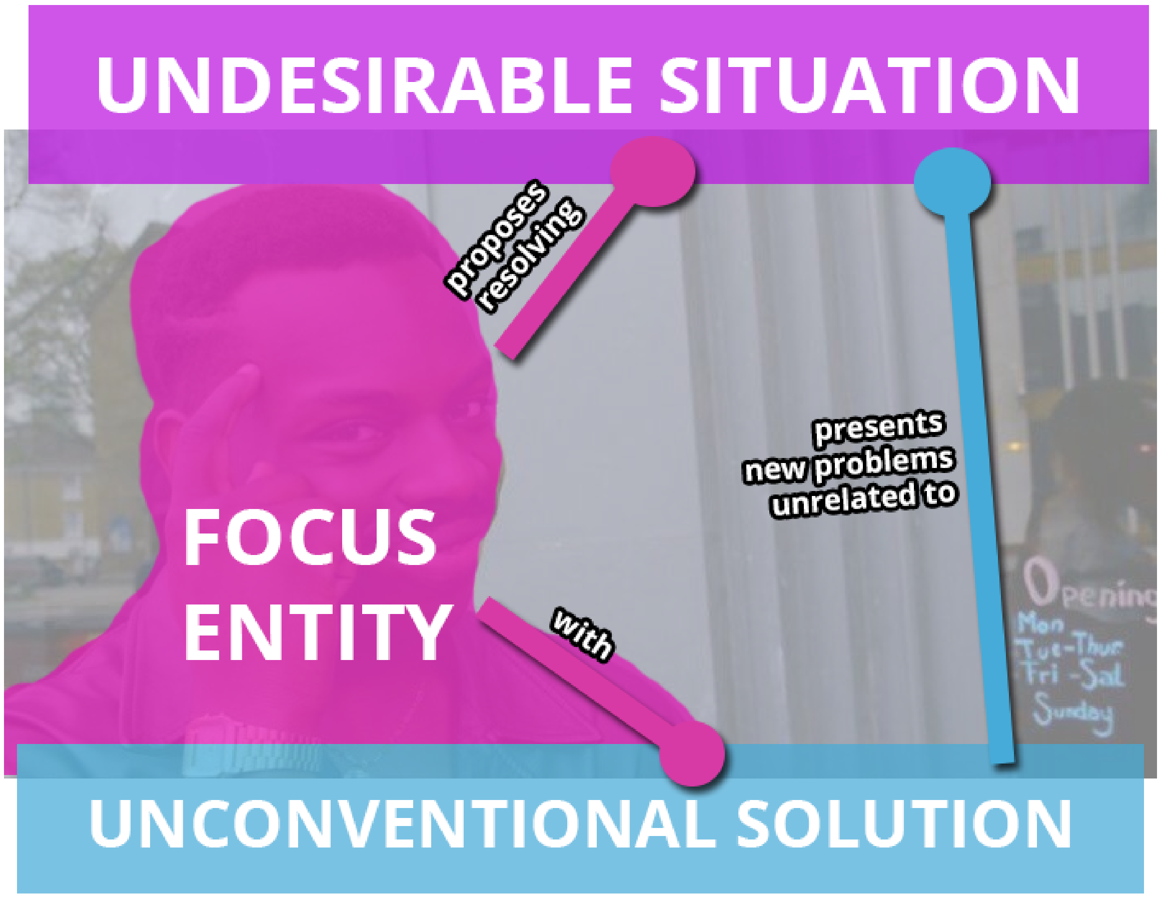


Figure 6.13: Breakdown: Relational Structure of "Rollsafe"

"Rollsafe" Relational Grammar

1. PROPOSES-RESOLVING (focus-entity, undesirable-situation)
2. WITH (focus-entity, unconventional-solution)
3. PRESENTS-NEW-PROBLEMS-UNRELATED-TO (unconventional solution, undesirable-situation)

"Rollsafe" Programming Meme Grammar

1. PROPOSES-RESOLVING (reader (IMPLIED), "Can't Get an Error Message")
2. WITH (reader (IMPLIED), "If It's Stuck in an Infinite Loop")

3. PRESENTS-NEW-PROBLEMS-UNRELATED-TO (“If It’s Stuck in an Infinite Loop”, “Can’t Get an Error Message”)

Evidence for “Rollsafe” Structure

Many participants clearly indicated that while the unconventional solution does solve the undesirable situation, it poses a new problem. Participants 8, 14, 15, and 28 each describe the details of why this solution is undesirable. Participants 9 and 24 indicate directly that the solution causes another problem, even without indicating the details of why. Participant 22 indicates a further relation that is generally recognized with this meme — that the humor is in the proposition of the unconventional solution, which comes across as seeming intelligent but is not due to the presentation of new problems. Several other participants indicated the phrase “big brain” in relation to this meme. This references another meme, the idea of someone doing something perceived as high intellect, when it is usually not.

> P15: Their code is working, but it’s in an infinite loop so it’s not working and that’s kind of the those two things playing against each other is the humor [...]

> P28: The idea is not really bright, but it is true. [...]

6.3.2 Humor, Self-Identification, and Sense of Belonging

A common theme aligning with existing research on memes was their ability to connect with emotions and experiences. Many participants indicated personal experience with the ideas several of the memes conveyed.

Evocative for many participants was the “Does Your Dog Bite?” programming meme. Participant 11 indicates clear experience with frustrations related to starting indexing at one, punctuated by “ahh”. Participant 12 indicates this would wound them as well, which Participant 20 echos in stating the meme hurts them a little bit. Participant 15 identifies themself as “Team Zero”, indicating belonging to a group represented by this meme. Participant 25 was incredibly vocal about their feelings regarding MatLab in relation to this meme.

> P20: This one hurts a little bit[...]

> P25: *sharp intake of air* Noooooooo *laughs* do you know that this is what MatLab does? *laughs* It’s so bad. [...]

Several participants also offered examples of their own experiences relative to the content presented in the memes. Participant 25 recognizes the noncompilation in the “Who Would Win?” meme as something they relate to from their own experiences. The debugging chain in “Gru’s Plan” is something Participant 29 has had experience with as well. Participants 6 and 10 both indicate that they have had experience with professors behaving in the exact way the “Whatcha Got There?” meme suggests regarding global variables.

> P25: [...]Well, uh at least to me, because I forget which curly bracket goes to another, so I’ll delete like a whole if statement and then like oop everything’s red and Java hates you now.

> P6:[...] *chuckles* yeah, professors always drill it into our heads like don't use global variables [...]

Participants also showed the ability to not only derive meaning from the relational structure, but also abstract that meaning to more applicable scenarios to their lives. In the non-programming variant of "Is This a Pigeon?", the subject was "refreshing Twitter every 2 minutes for an hour" and the 2nd-subject was "working on my book". Participants 8 and 10 indicate that the fact that "book" is directly indicated does not matter. The meme is about procrastination, and thus, relevant to any experience of procrastination. Participant 15 also references the ability to "self insert" into the meme based on its context, and still understand the meaning. They reference the non-programming variant of "Whatcha Got There?" and how despite not having experience with an illegal racecar, they can still imagine and contextualize themselves to that scenario based on the relational structures. Participants 25 and 26 go further in offering up examples of experiences from their lives they felt related to the racecar meme. Participant 25 indicates a similar scenario, though not with an illegal racecar, and is imagining how the critical entity must have felt based on their reaction as the focus entity. Participant 26 identifies a situation with their mother and clothing that is a completely different domain, yet relevant in the tonal reaction and behaviors that are part of the relational structure.

> P10: [...]just replace book with literally anything else and I think most people would abstract or they kind of get the point.

> P15: [...] memes don't need to necessarily apply to real life [...] you can figure out the meaning from the context provided

Participants also indicated many times they had interacted with the meme templates presented, showcasing relevance to their daily lives. Participant 24 indicated having seen an exact meme used in the study before. Participant 20 noted sending a meme to highlight their own emotions. Participant 16 identifies an example they saw that day about classes moving to online that used the shown template. Participant 12 even indicates participating in meme creation within the same week as the study was conducted. Participant 8 suggests the way they might use the meme in conversing with their peers, acknowledging the social and communicative value of memes. Almost all participants, even those who said they had little familiarity with memes, had some knowledge of the memes and their structures, and had seen or used at least some of them before. It was quite hard to recruit participants who had *no* experience with memes, despite actively trying to. This punctuates how relevant memes are to the cultural knowledge of our participants.

> P20: I know I've sent this to people [...]

Several indications were made (especially when showing the blank templates) that the context provided by the domain matters in *who fully understands* the meme. Participant 20 recognizes that memes exist on broad scales, but also within communities, and that this can affect who understands it. Participant 12 suggests the "mom index" (mentioned by other participants), which indicates that a meme should not have a high barrier to understanding. This suggests that participants do not feel memes should be "cryptic", but forms of communication that convey their feelings and ideas.

> P12: [...]if my mom can understand the meme, then it's a good meme

Participant 29 beautifully sums up the sense of community and belonging that memes can create during the post-interview. They note that memes on topics of interest — even academic topics — create a sense that there are other people "like me" out there, who enjoy the same things. Sharing humor and interests provide two strong bonding and engagement mechanisms. This participant suggests that programming memes allow a vehicle for both.

> P29: [...] I love programming memes I think they're so funny when they're done right. Um, and so it kind of like combines those two worlds [...] like ay I recognize that and it's talking about something I know about [...] the recognition of oh my gosh there are other people who are like me who they like memes and they like programming and they're making content about it.

6.3.3 Subversion and Career of Metaphor

Participants noted the ways that memes follow rules, but also that humor can lie in appropriately subverting them. Several of these comments came as an unexpected consequence of the programming meme for "Gru's Plan".

Typically, "Gru's Plan" is expected to contain the same information on the board in the third and fourth panel. However, in the programming meme, the board changed from two bugs to three bugs. This subversion was not planned in any part by the test design — in fact, it was not even recognized that participants may interpret it as subversive.

Participant 6 shares through laughter that the change makes the meme feel even more true, that it feels like the bug problem grows. Participant 9 also indicates that the problem is getting worse, and that this change works, but that it would not work if it were reversed from three bugs to two bugs. Participant 12 begins explaining the meme, only to realize in the midst of their explanation the panel change. They specifically state this subverts your understanding but in this scenario, that it adds to the comedy due to the bugs multiplying. Participant 15 states that in seeing the meme so much, you feel as though you can just "fill in" the fourth panel, which aligns with participant 12's behavior — and that changing it adds to the humor. Participant 25 expresses excitement and surprise at reading the meme and seeing it changed. Participant 28 identifies this as a joke not only about programming, but a joke at the template itself by changing how it behaves. Participant 29 describes how the modification may not have made sense given the usual sense of "realization" in the fourth panel, but that the growth of the problem allows it to still be relatable.

> P12: [...] *laughs* oops I didn't even read the last panel *laughs* it *laughs* it it's subverts your your understanding of the of the meme [...]

> P25: Oh nooo, oh so there's a bug, and you fix the bug, oh oh it changed! *laughs* Now we have two bugs, and look at it now there's three. *laughs* [...]

The above evidence from "Gru's Plan" led to additional discussion regarding the idea of subversion in memes. Participant 12 notes how subversion can "bring new life" to a meme and reinvigorate its humor. Participant 10 describes verbally while discussing the blank template for "Gru's Plan" an entirely separate subversion involving more panels. This subversion is intended to relay the idea that even though the plan diverged, the focus entity still makes it work.

Participant 21 noted the ability to “deep fry” the “Gru’s Plan” meme. They describe in detail what process deep frying requires. This is also a defined set of rules. They note that deep frying is a specific type of subversion that can heighten the apparent emotion of the meme.

> P21: [...] artificially put that effect into it to like show that there’s more emotion [...]

Participant 28 suggests the ways memes can “build” on each other as an act of subversion, by using multiple memes together in order to convey a more specific complex meaning.

> P28: [...] memes can build off of each other and can interact with each other [...] use multiple different templates to make like a single template [...] kind of mesh them together a little bit [...] just like brings both of those ideas together more, like one that’s even funnier

Participant 29 provides some interesting insights regarding the growth of a meme template. They describe how there’s no “written description” of how the meme works, it’s continual application of the same template by many people that makes the relational structure implicit in the image. They further add that when a meme image starts circulating, it isn’t a meme yet — it requires more use so that the relational structure becomes conventionalized within it.

> P29: [...] you see the meme used so many different ways in so many different jokes over and over again [...] you just kind of like subconsciously learn what each one means, you know? [...]

> P29: [...] a meme is something that’s used over and over again in the same format for different context [...] once it gains momentum and you see more people using it then you start to understand the context of it like oh this is a meme, I have more understanding of what it is now [...]

This evidence lends credence to applying the career of metaphor to memes. The images are at first novel, but become conventionalized as they spread and new domains are applied. Memes may also enter the frozen stage, in which the context they originated from is still remembered, but they are not largely referred to by their abstract relational structure. Finally, memes can become “dead” (“dead meme” is in fact a statement that was said by participants). A dead meme, much like the career of metaphor, has been used so much that it is often no longer considered humorous by the general population. The meme has “become” the relational structure it represents, which removes the humor it once possessed.

The capability for a meme to become ‘dead metaphor” points to the intrigue of subversion. As Participant 12 noted, it allows “new life” to be breathed into the meme. One must now reason about why the subversion was chosen, and what it adds to the existing structure. The fact that participants appear to enjoy when known memes do this suggests they are not pure relational vehicles. They are expressive communication tools that participants want to engage with, reason about, and enjoy.

6.4 Implications for Pedagogical Use of Memes

Several questions of interest were considered through this investigation.

Do certain forms of analogy enhance engagement and understanding when learning programming? Using memes as a relational analogy tool produced several instances of clear engagement and enjoyment. Many participants also reflected on their own experiences in a way that may increase sense of belonging as a programmer.

Participants in this study were not required to be learning programming. However, several participants without prior knowledge did attempt to formulate ideas through the interview and were able to draw reasonable conclusions about the relationships presented, even without any contextual knowledge. When used in a pedagogical environment where the topical information is actively presented, this ability to identify relations may help encode relevant associations and deeper understanding about the topic.

Can students draw meaning about pedagogical content knowledge in computer science from internet memes?

Even without prior content knowledge, participants can draw meaning to a degree. Participants created associations between the ideas based on the relational structure of the meme, and reached some conclusions on the programming content.

Interestingly, several responses with misaligned ideas or indications of lacking content knowledge drew parallels to participants *with* content knowledge. This suggests that knowledge of the relations promotes reasoning to understand them, even when context is lacking. In a classroom seeking to teach this context, learners may achieve greater understanding and assimilation of the concepts through this process.

Can memes convey information in a way that could provide value in a pedagogical setting?

This evidence suggests there is certainly observable value to the use of memes. Engagement, relevance, and sense of belonging are three major values that the use of memes can promote.

A recommendation for using memes in the classroom would be following the same best practices of general analogy use. Ensure the meme communicates a specific idea, and contextually position its use. Encourage analysis of what the meme is suggesting, and how that relates to pedagogical ideas. Investigating relevant programming memes can be an engaging classroom active learning discussion. It can also be implemented as a creative exercise where learners design a programming meme and elaborate on how it conveys pedagogical content knowledge.

6.4.1 Considerations

This study was conducted within our university. Our recruitment was through word of mouth so it likely does not represent our entire university's population. Many participants had a strong sense of understanding memes, and several had at least some background programming knowledge.

Many participants with programming knowledge possessed C-family language knowledge. At least two memes specifically reference C-family style syntax. However, there were participants with no C-family knowledge who were still able to draw conclusions that paralleled those with knowledge.

Finally, it is important to remember that not all learners may have background or interest in memes. Learner interests are varied, and memes may not motivate engagement for all learners. Within this study, not learner stated they had absolutely no familiarity with memes, despite active recruitment for participants fitting this criteria. This appears to exemplify the cultural prominence of memes for our learners at time of writing — memes are integrated in many ways to their cultural context. However, this does not dismiss the concern that being disinterested in memes or lacking understanding of them could cause learners to feel marginalized. This consideration highlights the value of additional investigation of active learning discussion as a classroom: working together to understand the meme and its implications may allow learners without knowledge of memes a space to learn from and alongside their peers, without needing to understand the memes prior.

Chapter 7

Prisms: Case Studies of CS Learners

In closing, I have conducted interviews with our CS1 learners in order to showcase their diverse experiences and perspectives. This is followed by an analysis of some thoughts on analogy from learners in an upper-level course, providing perspectives from more experienced learners.

Before diving into the interviews, let us first ensure we broaden our lens a bit — the learner has many more factors to consider than *just* the cognitive process of learning, after all.

7.1 Holistic Considerations for CS1 Perspectives

Environments and the elements in them play a role in our perception and processing of information. Designing an educational experience that centers the learner requires understanding all the factors that can affect that learner: the "whole system" of the computing classroom. Exploration of all of these factors are *not* the focus of this dissertation. In order to pivot to our learners' perspectives, however, it is important to consider how many additional factors can influence their experience. Factors to consider regarding the classroom and a learner's experience with it include:

- Sense of Belonging [39, 115, 128, 158, 160].
- Perceptions of the Discipline [21, 59, 123].
- Sociocultural Factors [23, 123, 132, 160]
- Communication with Classroom Community [21, 61, 81, 97, 132, 152, 161]
- Active or Group Learning Dynamics [21, 25, 90, 150, 151]
- Learner's Intent in Taking a Course [15]
- Beliefs Surrounding Pedagogical Tools [4, 146]
- Interpretation of a Pedagogical Tool's Use [4, 20, 146, 165]
- Ordering of Topics in the Course [168]
- Classroom's Spatial Design [8, 89]

7.2 CS1 Guided Interviews

Guided interviews were conducted in late Spring 2020 with CS1 learners, as well as learners who had completed CS1 the prior semester. This was done in order to obtain perspective on their thoughts and feelings surrounding the course.

7.2.1 Interview Study Design

The general guiding structure of the interviews can be found in Appendix I. The guided process was dialectic with clarifying or summarizing questions incorporated, in line with the approach used in Chapter 6's study. This allowed for better interpretation of the participant's intention, and promoted responses due to feeling less "clinical" as a procedure.

A timestamp notation approach was used just as in Chapter 6, based on Brown's suggestion [10]. This allowed "selection before selection", reducing bias by notating interesting phenomena prior to any investigation of their meaning. All interviews were approximately an hour, and transcription was pinpointed to these timestamps to reduce the initial analysis space.

7.2.2 Data Collection and Analysis

In transcription of participant timestamps, some were collapsed together due to ongoing dialogue that relevantly connected them. Of the 9 participants, 383 entries in total were transcribed.

Each learner's interview is analyzed as their own vignette. In reviewing the timestamped data, each learner's vignette presents a personal "major" theme they conveyed through the interview, as well as indications of "overarching" themes across the interviews.

Full quotes that are referenced in the compilation of the vignettes are included in Appendix J.

7.3 CS1 — Student Perspectives

Participant 1

Participant 1's journey in CS1 was quite distinct. They begin by identifying past experiences with code across different languages. Participant 1 suggests difficulty in getting these to work, and in finding "quality information sources". They share that they enjoyed learning what each piece of the code does, and found this important.

Participant 1 then elaborates on some of their difficulties as a non-traditional student. They draw a contrast between the military and the classroom, expressing how "bare bones" instructions from the military can feel and how the classroom is rich in perspectives by comparison

They continue by sharing one of the largest obstacles with the course: difficulty remembering information, which caused immense frustration. While other students may not experience the same exact difficulties that participant 1 indicates, cognitive overload is a theme any learner can relate to. Participant 1 does an admirable job of conveying the difficulty learners can have with congitive load. The participant indicates how it affects them, especially when they "know they should know".

> P1: [...] I get tripped up really easily and I get confused and frustrated really fast so like I'll be tracking an issue and then all of a sudden something interrupts me and now I I don't even remember what's going on and then I start getting more and more upset and confused and frustrated [...]

Participant 1 goes on to indicate the difficulty that complex programs cause as a result. They describe the garden lab program, which had three Java classes interacting with each other. The participant notes that movement between these classes exacerbated difficulties. Participant 1 is already working with difficult cognitive pressure in building single class programs with multiple parts. Moving across Java classes where they can't even see the full code creates additional pressure on them. They also indicate heavy reliance on physical resources such as the book, as they allow more control and are easier to review.

When asked what they were most proud of from the course, they indicate their resilience. They also state confidence in becoming more of a "conversational coder".

> P1: [...] it's like being able to read the newspaper in another language, where you're like yeah, I can I can understand, I get the gist of what's going on here, but I may not be able to understand every, I can't, translate every word here, but I I kind of understand what's going on with the code, just like you would if you were learning an actual language and trying to you know, read another country's newspaper.

Participant 1 also notes joy in seeing connections between ideas: not just how to write a variable, but how this allocates memory, and that memory allocation physically happens in the computer. They express happiness in knowing not just *what* to write, but why writing that does something.

Continuing to describe their cognitive pressures, they indicate that overload comes from a multitude of factors. They may start with best intentions, but reach a "breaking point" in which they need to just move on.

> P1: [...] there's too many emotions and signals kind of coming in at you to kind of do it and that's, that's a lot of what happens a lot of times and it's like it gets to the point where it's like I can't do this anymore and I just gotta turn in something [...]

They vividly describe the difficulty of inert knowledge: knowing something but being unable to do it.

> P1: [...] it's like going to pick up a fork, you know, you don't think about it, you just do it, and then all of a sudden you just can't do it, you know. And it's like, what's up? Where it's like, I don't know if you've ever been paralyzed before but its *chuckles* you wake up and you can't move and you're like what, but that's what it's like [...]

I asked the participant if they remembered rubber ducks at all when we discussed the topic of objects. Given the participant noted recall difficulties, I wanted to prompt to see if this was memorable to the participant. What was interesting was not that the participant remembered the ducks, but what came *after* that. Participant 1 exemplifies networks of connected knowledge. That same day we described how the class-object process is like a factory assembly line, and the ducks were specifically part of a "Duck Making Factory" demonstration. The participant shares a cascade of connected ideas after the single prompt, indicating these ideas became interconnected in their conception of objects.

Participant 1 ends by stating that overall, they had an amazing experience with the course despite the obstacles. They indicate that they didn't feel difficulty stemmed from a lack of help available, but from an inability to feel they were in a space to receive it.

Participant 2

Participant 2 indicates some of the most difficult programming aspects are "the little things". They also note the way that active learning and pair programming helped them by providing a resource in their fellow classmates. Participant 2 indicate that not only did they benefit from directly talking to other students, but also from hearing other students' questions and the associated answers.

> P2: Probably the most useful thing is being able to bounce ideas off of other people, I think that environment's really great.[...]

They go on to note that they recognize that communication is not only a major value for their learning process, but a skill for the workforce. They indicate experience in the workforce that has helped them recognize the necessity for communication skills.

When discussing what they might have done differently, participant 2 indicates they would have changed their work management approach, but suggest they are improving at this. Within the CS1 classroom, many students are just beginning their college careers, and likely learning to manage workloads just as participant 2 suggests.

Participant 2 shares that the enthusiastic nature of teaching made the class interesting. They allude to the peanut butter and jelly demonstration on the first day of lecture as an example of this, and that it ranked as one of their most engaging classroom experiences. The participant also indicates that they now feel they have developed a general association of programming with ducks.

In considering if any course topics caused strong emotions, they note that a personal connection is the only real way this becomes elicited for them. However, they specifically note a small assignment which asked them to think of a mentor. Clearly, this student completed the task and felt a very personal connection to it. Participant 2 goes on to note that the course was only needed as part of their degree completion. They had no idea what they wanted or expected to get out of it, and indicate that they feel the course exceeded their expectation.

> P2: Only needed this course just so that I could get a specific credit on my degree. I didn't know what I wanted out of the course [...] I have definitely gotten more than what I expected [...]

Participant 4

Participant 4 indicates that the peanut butter and jelly demonstration was humorous. When asked for further details, they suggest it helped them recognize the need for specificity when programming.

> P4: [...] you can't just make assumptions and the computer knows what you want it to do

This participant had particular affinity for one of their pre-CS1 projects, a Rubik's cube solver. When asked what they were most proud of in CS1, they indicated this project, discussing it several times throughout the interview. When asked what specifically from *CS1* they felt proud of, they indicated being proud of what they knew was a "monstrous" nested loop, because it was "a way to to do it" and that allowed them to complete the lab problem within the two-hour lab time. They continue throughout the interview describing how they connected with concepts in CS1 by often relating them to the Rubik's cube solver, such as how understanding objects could have helped them write more concise code.

> P4: Probably would have made an object called like CubeTurner or something had a constructor that could be put in some parameters to have it act on the different sides, and then it instead of having ten bazillion lines of code [...]

They also indicate that demonstrations like the ducks help, because they make things less abstract.

Finally, in walking through a homework problem, participant 4 observes their solution "matches the template", suggesting pattern recognition of what a code piece for that topic "should look like". They also note using the IDE as helpful in validating their work.

Participant 5

Participant 5 states they believe the instructor creates strong analogies, and these are helpful with novel topics. When asked for clarification, they indicate elements within the lecture slides, but also the physical rubber ducks.

Participant 5 also indicates the priority of completing assigned work. This need to complete their work can be at odds with their learning process. They note the need to "get it done" and how this can become the top concern. When asked what they might do differently, the participant suggested they would go to lecture more. I ensured the participant was not just saying this because they were speaking to the instructor, and they clarified by adding to their previous thoughts on prioritization. In trying to complete homework they would often skip classes, and doing so made them feel more lost in the homework.

> P5: [...]then I'd be like oh, I missed this week's both of the lectures this week, and I don't know exactly what's going on with this, so then I would have to spend even more time like going through the uh lecture slides and trying to teach myself.

Due to restructuring of CS1 course's materials, some book chapters were moved out of order. This participant indicated that the topics felt well-connected, and the "jumping around" was not a problem.

This participant did not have any idea of how programming would be applicable to their future career or aspirations, outside of the development of personal applications. They shared ideas they were already considering of what they would create.

While discussing arrays, the participant indicated their syntax "made sense". Arrays when initialized can have their size indicated by a square-bracketed number, but a square-bracketed number is also used to indicate index. The participant states it is not difficult to figure out the syntax's meaning due to its context.

Finally, participant 5 suggests that in their assignments, they could have been more specific to understand the topics better. This reflects back to the idea of "just doing it" they had noted prior. They feel taking more time to explain would ensure they really understood what they were writing.

> P5: [...] if you have to explain it then that in itself requires a certain level or a certain ability to know what's going on [...]

Participant 6

Participant 6 indicates their interest in not just programming, but in what happens within the computer hardware when programming.

> P6: [...] what you're actually doing and like memory allocation, stuff like that, so I definitely take more of a hardware approach than um, probably like computer science majors [...]

This participant also suggests they are interested in "authentic" applications of programming, like what one might experience in industry. They indicate programs like the garden lab make sense, but don't seem to translate to the type of projects one might create in a workplace. Later in the interview, the participant comes back to the garden program, stating that while they may have been critical, they felt its relationship to the real world was helpful. They also draw upon an example from the slides of HockeyPlayer objects being created within a Rink class.

> P6: [...] I know I just totally ripped on the garden produce bunny thing but that uh it it really does like kind of when you relate it to something in real life like that it does, I think help me you know kind of grasp like the big overarching concepts of it [...]

Interestingly, this participant indicates that they feel like Java is not "meant for" arrays, because the size of them cannot be changed after initialization. What is striking about this is the participant's prior indication of interest in hardware processes underlying programming. There is a disconnect regarding the memory allocation process when it comes to arrays — something that is discussed in the course. Despite an interest in underlying processes, the participant believes that arrays should be able to change size. This seems grounded in existing knowledge of other programming languages.

Participant 8

Participant 8 notes that they find programming to be creative and magical, as they can bring their ideas to life. They also indicate an interest in the multi-object simulation projects, describing them as teamwork systems. Further, they describe liking animals as being a reason they are interested in the garden program, and also note several related concepts they must utilize to complete it.

> P8: [...] the uh rabbit one, uh and uh also the duck one because we need to uh, fetch constructors uh setters, getters, and toString method [...]

The participant indicates the rubber ducks being memorable, and connects two different demonstrations using the ducks. They relate both aliasing and constructors, which were two distinct uses of the ducks in describing reference types and specifically, objects.

Participant 8 also notes confusion at first that "noun does verb", believing the noun be a class name, not the object of the class. They indicate that through further investigation, they arrived at the correct syntax.

Participant 9

Participant 9 indicates objects feel like a key topic in understanding programming. This participant indicated prior experience with MatLab. They appear to suggest objects reshaped their entire notion of programming, but could be indicating programming in Java specifically. This participant relays their experience with MatLab, indicating they enjoy Java more. They state heightened confidence as prior to the course their experience with MatLab had been negative.

When asked about expectations of the course at the start, the participant indicated what most instructors know to be true: they did not read the syllabus. Asked about assignments they enjoyed, participant 9 indicates the creature program, where they created a Creature class and spawned Creature objects into a Conservatory. They also note the text prediction "Mad Libs" style program, which felt like something someone else might be able to use.

> P9: [...] it felt like we actually like made something [...] like you could use so it felt like I don't know, I I did something *chuckles*

Participant 9 suggests having more exploration beyond "just syntax" was helpful. They specifically indicate the T-Rex activity from lecture, and how it helped relate objects and classes to the world. They also indicate acts like naming the rubber ducks helps clarify their relationship of a common class, but that they are distinct objects.

In class, we also often did whiteboard activities where participants worked in small groups with whiteboards on a problem, then shared answers. The participant suggests that the structure of these activities — discuss the topic a bit, break into small groups, and come back together for further discussion, was helpful. They note writing code on a whiteboard was different than typing it into a computer.

> P9: [...] it made me understand like, instead of just knowing like what characters to type in the computer like it made me understand why I was typing em and it made it like a lot easier to apply it

The participant also draws a correlation to the concept of static and dynamically typed languages, which was covered in week one. They indicate that having a greater understanding of these ideas is powerful, as it should remove some of the difficulty of getting started with a new language. Participant 9 suggests that they feel happy that they could write a Java program if someone asked them, and are proud of that confidence.

> P9: [...] I had never felt like super confident with programming but I definitely if somebody like asked me to write a program in Java I'd feel like I could, which makes me happy. *chuckles*

They go on to suggest that they started the class with low expectations, and it has been transformative for them. As a non-major, they thought it would be a class they did not commit much effort to, but ended up enjoying it and putting it at the forefront. They are now considering further computer science courses, and even an advanced degree in CS.

Participant 9 provides a distinct perspective on work allowing unlimited tries. Our CS1 course allows unlimited tries on quizzes to provide a space for mastery learning. The participant indicates this became a "ritual" activity, where they would simply "bank" correct responses until they felt satisfied with their score. They note even though other work may have been tedious or difficult, it made them apply their knowledge more. This participant also observes during a homework assignment discussion that the classroom "mantra" of variable values had an impact. A homework question asked what the variable's value was, and they note that this type of question strongly correlated to the phrase "location in memory". This shows an application of classroom cultural norms — this question was asked during lecture frequently when discussing reference types.

When asked about their initial thoughts on objects, Participant 9 describes fear that this would be the topic they did not understand. They note the duck activity create an "aha" moment for them regarding instances. They also note frustration with getters and setters, wanting to just obtain and edit the variable details directly from the object.

> P9: [...] so if you're like oh Jonathon is like an instance of like the Duck class like that would make sense, because there's like there can be like all kinds of different ducks, but like, Jonathon's one of them so like, when you described it that way, like it made sense. Which was like, also kind of an aha moment [...]

Participant 9 indicates that "real life" grounded examples were impactful to their understanding. They note that industrial applications, like a machine and its parts, would have been confusing. However, they already understand ideas like ducks in ponds.

The participant relays difficulty with finding resources that can help online. They indicate that examples or suggestions provided online are highly contextualized, making them difficult to apply to the participant's context. Despite the amount of programming resources online, finding things that are "actually helpful" in the way they would like to learn is not easy.

They note that objects made them feel excited and rewarded, and that the ways objects can work together allows them to "make your own little thing".

> P9: [...]it was like really rewarding to get it to work, especially like the I don't know like, the conservatory and like some of the ones that were like, bigger and you made like a whole scene, was really fun and you're like oh look, like, I have my Duck pond or like, I don't know, you like made your own little thing [...]

7.4 After CS1 — Student Perspectives

Participant 3

Participant 3 recalls the garden program and that they found it fun. They suggest one has an intuition about how the problem should work, and that this makes it easier to focus on the programming part of the problem, rather than reasoning about how the problem space worked.

> P3: Honestly that bunny program like just *chuckles* there's just something kind of fun and you know, it was a more interesting than if I was *pause* makin' files open or something weird. [...]

They further go on to state that the multi-object systems felt like "my own little zoo", and that this was fun to build and interact with. Participant 3 recalls feeling frustrated when their peers seemed to "get it" faster than they did. This may have discouraged the student from feeling as though they "belonged" at times. This participant also shares that they go back and review the slides from CS1 in their current course. They note that not only do the slides assist them, but they elicit emotion as well.

> P3:[...] Go back and like, read em and laugh. I mean I still go back and look at em. [...]

Recalling the rubber ducks, Participant 3 indicates enjoying the way this presented objects intuitively before coding them.

> P3: I think of your duck, your week 8 ducks. Like that's one of the first things that comes to my mind [...]

They also recall being a participant in the eraser tossing exception activity. They note using that example in their current course to help them reason through situations regarding exceptions. They observe that they started the course only needing it as a prerequisite, but through learning the material, they started getting more out of it and wanting to get more out of it.

This participant notes that the change in ordering from the book's material was helpful, as understanding objects earlier helped them understand other concepts more easily. They also distinctly remembers a "lightbulb" moment when they understood that the Scanner they had been using since first learning Console I/O was an Object.

> P3: [...] when you said Scanner is an object, like Scanner was a, like that all of a sudden just weirdly clicked, it was this huge lightbulb moment.

Participant 7

Participant 7 indicates that programming is an art, taking creativity. They also share gaining a lot of respect for the discipline as a result. They go on to describe the creature program being one of their favorites, indicating that the topic of objects started making programming make sense.

> P7: [...] I think that was that's when everything came together for me, and that was like my favorite part I don't know, that's just when programming started to make sense [...]

Participant 7 is candid about the way the semester can wear down a student. They indicate not really recalling anything from the course toward the end, but also having that difficulty in all of their classes. They also note that the book was overwhelming.

They indicate enjoying the peanut butter and jelly presentation, and appreciating when we would build and run programs together in class. Participant 7 shares that while they feel strong with concepts, syntax can be the most difficult aspect to get right. Connectivity issues in the Hangouts call prompted me to ask about the ducks, which the participant indicates they had stated during the connectivity issue as a memorable aspect of the course. They also are uncertain but recall "things being thrown" — this did happen with most notably erasers and ducks.

This participant indicates that their biggest takeaway was better understanding of what it means to program. They suggest they originally thought it was just "typing", but recognize now it requires creativity and critical thinking.

Participant 7 notes difficulty with transitioning to their next course due to being worn down at the end of their CS1 semester. They suggest "it doesn't stop", in reference to how programming knowledge continues to build on prior ideas. No matter what order we teach programming concepts, the complexity of continually adding new syntax can always be a struggle. They also indicated the end of semester slog in CS1 caused a negative mindset leading into their next course, which they pin on themselves.

> P7: [...] I went into 1122 kind of discouraged [...] Kind of made me scared for 1122 I was like shoot like am I way behind? I don't know. [...]

Participant 9 and I had an interesting discussion on "burnout", and they noted additional fear for their next course as well. They reference difficulty in the online transition during COVID-19, and that with a summer between their next course they are worried. Despite this sense, they know they'll "get through", showing a strong level of resilience in spite of the obstacles they perceive.

Participant 7 also discusses the garden program, and how communicating with their partner allowed the ideas to come together in a way that aided them moving forward. The participant shares that despite not even fully remembering programs like the creatures, you could "just feel" that they made sense.

> P7: [...] same thing with the other program that I'm obsessed with, like the creatures, like everything just kind of like it just has this, you can feel it, like when it makes sense, I don't know what it was [...]

This participant ends with suggesting they had set low expectations, but managed to surprise themselves in the course.

7.5 Discussion of Case Studies

Each of the above case studies centered around conveying a major theme observed in the data. In compiling the case studies, I was not certain how I might characterize the themes, only that I could tell they were present. Doing so, I recognize here that each major theme is associated to aspects that have been investigated through this research.

- Participant 1: **Cognitive Load.** The difficulties and frustrations that a learner can feel as they work to apply knowledge they know they have learned.

- Participant 2: **Communication.** The importance of conveying information in a way that others can understand and reason about.
- Participant 4: **Personal Connection.** The value of seeing something you are personally invested in as a pedagogical bridge.
- Participant 5: **Motivation.** The ways in which the context one exists in can modify how they approach a problem or course.
- Participant 6: **Perceptions.** The way notions of relevance and utility can impact the way material is viewed.
- Participant 8: **Topical Interest.** The enjoyment of materials can increase engagement and memorability.
- Participant 9: **Sense of Belonging.** The finding of one's identity within a discipline can modify their perceptions.
- Participant 3: **Engagement.** The application of oneself to the learning process aids in further retrieval and benefits.
- Participant 7: **Resilience.** The obstacles a learner encounters may be great, but their ability to overcome them is often greater.

The participant vignettes also encapsulated several overarching themes across many of them.

- Physical activities and demonstrations that are analogies to programming concepts were not only memorable, but often triggered recall of the concepts they were analogies for.
- Programming problems that are analogies to "real world" concepts, such as the garden problem "make sense" to these learners, allowing them to reason about the programming ideas instead of the problem space itself.
- Objects made programming "make sense" for many participants. Participants indicate programs with this focus as being among their favorites. They also indicate many moments of clarity as they came to understand the topic more.
- Participants enjoyed learning more than "just syntax". They highlighted in several instances that coming to understand the *why* instilled a sense of greater overall understanding.
- Learners often do not know what to expect from CS1. Many end up surprising themselves with what they took away or how well they did, and almost every learner indicated enjoying their time in CS1.

7.6 Relationships with Analogy Beyond CS1

A substantial portion of CS Education research focuses on CS1. In closing this work, I would like to not only look at CS1, but begin bridging to courses beyond it.

7.6.1 Survey Procedure

The instructor for our concurrent computing course administered a survey to learners in the course for extra credit. The questions on the survey can be seen in Appendix K. Of those who completed the survey, 55 respondents consented to be part of this research and their data was collected.

While further analysis will be conducted, preliminary observation of learners beyond CS1 appeared a fitting "look ahead" as we close out our journey in this work.

One question asked of our concurrent computing students that is fitting to close out our investigation on analogy was:

> Has your perception or use of analogy changed as an upper-level student? Do you feel you use them more/less, or that they are more/less effective? What do you feel may have impacted any changes (or lack thereof) in your perception?

Each participant's full response to this question can be found in Appendix L.

7.6.2 Analysis

While our upper level students vary in feelings and experiences with analogy, there appeared to be overarching themes. I conducted a preliminary grounded theory analysis around how learners appeared to characterize analogy in their response. Responses were analyzed to identify how often learners used analogy and how their use changed.

If a learner indicated they use analogy at some level, additional themes were not marked for them as this was the target theme. Multiple themes were *only* marked for a single learner when they appeared to have equal prominence in the response — otherwise, the most dominant theme was selected to categorize their response. Multiple themes typically only appeared in learners who did not directly indicate a response for their opinions on analogy use through their studies. Only one response could not be identified for any of the themes below, and was not tabulated due to its lack of any analogical theme.

7.6.3 Observations

Overall, upper level students felt analogy is more beneficial to them and that they use it more now, with 27 responses being identified as relating to this theme.

Many of the responses that indicated that analogy can be useful and valuable centered on the learner's use of analogy to explain concepts to *others*. This shows use of analogy and perceived value in it, even if these learners often did not indicate analogy being used to help *them* learn.

Learners also expressed concerns about the need for "good" analogy, and trepidation about inappropriate analogy use. This concern aligns with the work we have done in understanding ways to develop well-formed analogy.

Table 7.1: Preliminary Analysis of Analogy Use Themes for Concurrent Students

Emergent Theme	**# of Responses**
More Useful in Upper Level, Using More Now	27
Same Use Across Levels, Consistent Value	6
Can be Useful/Valuable (Especially in Explaining)	6
Be Careful; Effectiveness Needs to Be Considered	5
Less Useful in Upper Level, Using Less Now	4
Same Use Across Levels, Value Depends	3
Don't Need Them in Upper Level but they are Appreciated; Lower Level Has More Value	2
Don't Help at all	2
Wish Analogy Had Been Used More Prior	1
Analogy is Used Differently Now	1
Difficult to Make Analogies, but Useful	1

7.6.4 Implications

While the CS Education research community argues against the use of analogy, there is clear perspective from our concurrent students that analogy certainly has value in learning their course's material. As we close, this adds to our evidence of analogy's value in computer science. Here, we close with seeing the potential of analogy — we can not only increase its value for our introductory students, but look beyond.

By exploring analogy design in courses across our curricula, we may be able to promote stronger reasoning and relational schema development. This extends from our beginning misconceptions on syntax and concepts in CS1 to the design of entire upper-level systems and processes.

Chapter 8

Dispersion: Conclusion

This dissertation explored two major research questions, with the second being in response to findings from the first.

The first major question was: *What factors influence how novices learn programming?* Exploring mental models, and cognition, this research provided a wealth of insight into this question from across disciplines. Case studies of learners exemplified their own learning journey, providing further insight and consideration. Additional factors beyond *the process of learning* provide evidence for considering the classroom as a "whole system" of factors culminating as the learning experience. Among these factors are learner engagement and relevance. These factors, coupled with cognitive evidence, encourage consideration of analogy in the learning environment. In exploring this consideration in computing education research, trepidation surrounding the use of analogy was evident. Investigation of this trepidation highlighted valid concerns, but no concerns that justified discouraging analogy use in computing education when compared to other disciplines.

The second major question addressed: *How can analogy be more appropriately leveraged in programming education?* is answered with my analogy design tool, OPAL. This tool is tested and its value in promoting well-formed analogy design and critical analysis of analogy within programming is exemplified. Additional forms of analogy are explored, such as physical analogical representations and Internet memes. The value of memes as pedagogical tools is explored by utilizing programming content in meme structures. Evidence suggests learners *literally* interpret memes as the relational structures they represent. This allows them to be a strong representational vehicle for any content, including pedagogical content, and can promote their value within the classroom as an engaging analogical reasoning tool.

The start of this dissertation raised the question: *are there learners who "just get" introductory programming, versus those that don't?* Throughout this research, mental models, the cognitive processes required of learning, and analogy's role within these were explored. The evidence in these chapters suggests that what might be perceived as "just getting it" is likely a general schema developed from prior experiences, which the participant is able to more easily map to the act of programming through analogical reasoning. Activating relevant prior conceptions of *any* learner can help them reason about and assimilate new knowledge. Analogy is shown to be a viable and powerful tool in promoting this connection.

8.1 Contributions

1. **A mapping prompting consideration of CS Education as an experience design problem.** Educational design is an experience design, as our pedagogical approach is one way learners come to experience programming. Recognizing the association between these fields

allows impacts and implications from experience design to influence methods used in education research. While care must be taken in not overextending this mapping lest commoditization or oversimplification of education be risked, this mapping can inform the exploration of research themes and methods that may be useful in designing educational experiences. Mental models and information mapping via analogical reasoning springboard such considerations in this dissertation.

2. **Extensive literature review from a variety of disciplines in order to advance understanding of novice programmer mental models.** Analysis from several domains on cognition is positioned in consideration of learning CS. This provides significant value to Computing Education by advancing knowledge assimilated in our discipline around "how learners learn". This literature review also highlights analogical reasoning and through it analogy as core cognitive processes.

3. **Work towards a theory of "expert programmer intuition" based in compiled cognition literature.** I posit on ideas related to intuition that are based in mental models and learning literature for this process, which centralizes abstraction via analogical reasoning and high activation of existing models. The concept of "intuition" and how programmers transition from novice-to-expert is a subject of exploration within computing education research, and this provides a contribution to that space.

4. **An investigation into the use of analogy in education to promote its use in computing education.** I highlight the value of analogy in education across STEM disciplines and ways to ensure positive analogy application. This investigation disarms the dismissal of analogy that exists within computing education research. Dispelling the stigma surrounding analogy opens significant further research potential in the computing education research community.

5. **An analogy design tool that facilitates well-structured analogy development, allowing for greater evaluation and critique of instructional analogy design.** I set groundwork for the value in flinging open analogy's doors by providing a tool to promote well-formed analogy. This tool aids in addressing concerns surrounding analogy's use by promoting structural and contextual considerations, and can uniquely promote multiple domain generation. I further contribute by testing this tool in the development of 55 CS1 analogies, showcase exemplary cases where this tool allowed for necessary critical analysis, and obtain feedback and input on the tool's use from other CS educators. This tool marks a paving stone in promoting future analogy work and beginning positive discussions and analysis of analogy within computing education research.

6. **Evidence of Internet memes as analogical relation structures that may provide value as a pedagogical tool.** I provide further consideration for novel uses of analogy by conducting a study which exemplifies the relational structure of internet memes and the utility of this in reasoning . This study's evidence suggests that when used in the classroom with relevant pedagogical content, memes can promote analogical reasoning and learner engagement. Exploration of their relational structure promotes consideration for "well-formed" memes that convey the desired knowledge. This exploration of memes as a pedagogical analogical reasoning tool and exploration of their relational structure is novel to computing education research. To my knowledge, this may also be novel across broader education and analogy research.

7. **Perspectives from CS1 and higher-level learners providing insight to their experiences with programming material and highlighting analogy's value in learning**

programming. I exemplify the impacts of the classroom environment with vignette case studies that each identify an overarching theme for learners. These vignettes also indicate the memorable and valued nature of physical analogical representations, providing further contribution promoting analogy's value in computing education. Learners in courses beyond CS1 indicate increased perception of the value of analogy, contributing to the promotion of further exploration of this topic in CS1 and beyond.

8.2 Future Work

The contributions of this work have left significant future pathways for continued research and exploration. Even in writing this dissertation, I consistently found myself writing, thinking *Wouldn't that be interesting?*, and promptly having to relegate that away in order to stay focused. I am incredibly passionate about the work conducted in this dissertation, and I am excited to explore it in future research.

One fount that has not been completely tapped is the extensive qualitative data sets collected in this research. Further analysis and codification of these data sets will inevitably lead to additional findings and new research questions. I wished there was time (and space) to devote to further analysis of these here: the meme interviews, the learner case studies, and responses to the concurrent analogy survey. Revisiting and continuing to analyze this data is sure to promote new findings and questions.

The introduction of the OPAL tool within this work provides many unfinished paths and further questions. Designing and testing additional applications of OPAL is of course among my future work. I would like to get more instructors using OPAL in order to further investigate its usability, identify new research questions, and create further analogies and discussion of analogies with it. Testing of the application of OPAL analogies in the classroom is worth additional investigation as well. Testing of OPAL analogies in the classroom was initially planned for this dissertation, but ultimately removed from its scope. Studies testing OPAL analogies in the classroom should look for not only learning value, but also engagement gains, in line with Haglund's considerations for educational analogy use [67].

Investigating "complex concepts" such as recursion and appropriate analogies for these is one unexplored use of OPAL. With the perspectives and first attempt at this from Instructor 1 in Chapter 4, it seems OPAL may help through a decomposition of component processes, investigating appropriate structural analogies for each process. OPAL can be leveraged for this, and may provide new insight into considering analogy for these concepts. The structural requirements of OPAL can then be looked at systematically *across* analogies for a concept, in order to recognize if some source can in fact fit all entities and relations. To exemplify, OPAL might be used to validate the analogy *A stack is like a Pez Dispenser* by decomposing a stack's component functions — push, pop, peek — and investigating desired knowledge about each as procedures within OPAL. Investigating general structure for each procedure provides a collection of general structures that *all* must be satisfied by the processes of the Pez dispenser entity in order for it to be a reasonable analogy for the "complex concept" of a stack. Further investigation and analysis for this use of OPAL is of particular interest.

OPAL's procedure-based design also suggests that it promotes a storytelling structure. The area of storytelling and story structure as analogical reasoning and learning tools is of particular interest to me as a researcher, but was out of scope for further exploration in this work. Additional research into storytelling and analogical reasoning will undoubtedly lead to new research questions. As indicated in Chapter 3, storytelling can promote context generation and further engagement in listeners. I am curious how OPAL and its potential for encouraging story structure may intersect, and look forward to investigating this further.

Research surrounding additional analogy forms and how the contributions here intersect is also of merit. Physical analogical representations such as demonstrations and embodied activities may benefit from their components being analyzed within OPAL for structural soundness. Further exploration into engagement with these representations and new concepts and sources to draw from, as well as studying the effects of these in the classroom, would be a valuable addition to this work. Suh [148] and I have explored the intersection of designing concrete representational comics using OPAL, and are continuing to investigate additional applications, considerations, and study designs.

The findings on Internet memes as analogical reasoning tools can be further expanded on and applied in the pedagogical context. While I used pedagogical *content* in the study, the memes were not situated in a classroom *context* where one is learning about the ideas. I believe designing activities to promote learner engagement with memes as an analogical reasoning tool and analyzing the effect of these activities would be valuable and add to the contributions of this work.

Syntactic keyword polysemy and the metaphoric nature of computing representations is also worth exploration, but was ultimately out of the scope of this work. The keyword “for” as an example lacks contextual value in understanding the code’s intention. This can modify learner conceptions surrounding its meaning. Several other keywords and even processes within programming environments have multiple interpretations as well, which can impact perceptions of use. I began investigation into programming keyword lexicon and the potential for polysemous confusion. I would certainly enjoy continuing this research to further understanding of engagement with technology’s metaphoric representations can modify assumptions.

Finally, I would like to continue investigation in my broader research theme of *Education design as experience design*. This requires analyzing and researching additional environmental considerations within the classroom context, and attempting to better understand the “whole system” of the CS classroom and learners’ experiences with it. I believe applying user experience principles and practices can promote this research and new questions in novel ways, and further highlight the contribution of mapping my first consideration of CS Education as an experience design problem.

8.3 Closing Remarks

I hope reading this research has provided greater insight to the choice of analogy presented in the title and introduction: the stained glass of knowledge.

Mental models are each a stained glass piece — connected as ecosystems of models which form the mosaic of our collective understanding about the world. The ways in which these pieces connect are fostered through contextual retrieval, adaptations, and analogical reasoning. The use of analogy allows us to make connections across seemingly disjointed fragments, forming unique patterns of understanding. Our mental models are impacted by our perceptions, experiences, and beliefs — these are the ways the pieces are uniquely tinted and cut. We add pieces to this mosaic of collected knowledge and reconnect them — the shape of it ever-growing and morphing.

Learning to program adds pieces to this mosaic for each learner, morphing its form in some ways while others are preserved. I hope that in reading this work, you have found new considerations in how we might help learners configure and assimilate these fragments.

"I want to line the pieces up....yours, and mine." [38]

Bibliography

[1] John R. Anderson and Robin Jeffries. Novice lisp errors: Undetected losses of information from working memory. *Hum.-Comput. Interact.*, 1(2):107131, June 1985. ISSN 0737-0024. doi: 10.1207/s15327051hci0102_2.

[2] Mike Anderson. Marrying intelligence and cognition: A developmental view. *Cognition and Intelligence: Identifying the Mechanisms of the Mind*, pages 268–287, 01 2004. doi: 10.1017/CBO9780511607073.015.

[3] Maya Angelou. Still i rise. In *And Still I Rise: A Book of Poems*. Penguin Random House, 1978.

[4] Ian Arawjo. To write code: The cultural fabrication of programming notation and practice. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI 20, page 115, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450367080. doi: 10.1145/3313831.3376731. URL `https://doi.org/10.1145/3313831.3376731`.

[5] Albert Bandura. *Social foundations of thought and action: a social cognitive theory.* Prentice-Hall, 1986.

[6] Paul Bartha. Analogy and analogical reasoning. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 2019 edition, 2019.

[7] Callan Bentley. Using analogies to assess student learning. *Inquiry: The Journal of the Virginia Community Colleges*, 13, 2008. URL `https://commons.vccs.edu/inquiry/vol13/iss1/4`.

[8] Briana Bettin, Linda Ott, and Leo Ureel. More effective contextualization of cs education research: A pair-programming example. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '19, pages 182–188, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6895-7. doi: 10.1145/3304221.3319790. URL `http://doi.acm.org/10.1145/3304221.3319790`.

[9] Max Black. *Models and Metaphors: Studies in Language and Philosophy.* Cornell University Press, 1962. ISBN 9780801400414. URL `http://www.jstor.org/stable/10.7591/j.ctvr6971f`.

[10] Ann L. Brown. Design experiments: Theoretical and methodological challenges in creating complex interventions in classroom settings. *Journal of the Learning Sciences*, 2 (2):141–178, 1992. doi: 10.1207/s15327809jls0202_2. URL `https://doi.org/10.1207/s15327809jls0202_2`.

[11] Achilleas L. D. Buisman and Marko C. J. D. van Eekelen. Gamification in educational software development. In *Proceedings of the Computer Science Education Research Conference*,

CSERC 14, page 920, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450333474. doi: 10.1145/2691352.2691353. URL https://doi.org/10.1145/2691352.2691353.

[12] Jaime G. Carbonell. Learning by analogy: Formulating and generalizing plans from past experience. In Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 137–161. Springer Berlin Heidelberg, 1983. doi: 10.1007/978-3-662-12405-5_5.

[13] Carlos Mauricio Casta Daz. Defining and characterizing the concept of internet meme. *CES PsicologÃa*, 6:82 – 104, 12 2013. ISSN 2011-3080. URL http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S2011-30802013000200007&nrm=iso.

[14] Yam San Chee. Applying gentner's theory of analogy to the teaching of computer programming. *Int. J. Man-Mach. Stud.*, 38(3):347–368, March 1993. ISSN 0020-7373. doi: 10.1006/imms.1993.1016. URL http://dx.doi.org/10.1006/imms.1993.1016.

[15] Parmit K. Chilana, Rishabh Singh, and Philip J. Guo. Understanding conversational programmers: A perspective from the software industry. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI 16, page 14621472, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450333627. doi: 10.1145/2858036.2858323. URL https://doi.org/10.1145/2858036.2858323.

[16] John Clement. A conceptual model discussed by galileo and used intuitively by physics students. In Dedre Gentner and Albert Stevens, editors, *Mental Models*, chapter 14, pages 325–340. Lawrence Erlbaum Associates, Inc., 1983.

[17] John Clement, David E. Brown, and Aletta Zietsman. Not all preconceptions are misconceptions: finding anchoring conceptions for grounding instruction on students intuitions. *International Journal of Science Education*, 11(5):554–565, 1989. doi: 10.1080/0950069890110507. URL https://doi.org/10.1080/0950069890110507.

[18] John J. Clement. Expert novice similarities and instruction using analogies. *International Journal of Science Education*, 20(10):1271–1286, 1998. doi: 10.1080/0950069980201007. URL https://doi.org/10.1080/0950069980201007.

[19] John J. Clement, editor. *Methods Experts Use to Evaluate an Analogy Relation*, pages 47–56. Springer Netherlands, Dordrecht, 2008. ISBN 978-1-4020-6712-9. doi: 10.1007/978-1-4020-6712-9_4. URL https://doi.org/10.1007/978-1-4020-6712-9_4.

[20] T. R. Colburn and G. M. Shute. Metaphor in computer science. *Journal of Applied Logic*, 6 (4):526–533, 2008. doi: 10.1016/j.jal.2008.09.005.

[21] Katelyn M. Cooper and Sara E. Brownell. Coming out in class: Challenges and benefits of active learning in a biology classroom for lgbtqia students. *CBE-Life Sciences Education*, 15 (3):ar37, 2016. doi: 10.1187/cbe.16-01-0074. PMID: 27543636.

[22] Stephen Cooper, Wanda Dann, and Randy Pausch. Teaching objects-first in introductory computer science. *SIGCSE Bull.*, 35(1):191195, January 2003. ISSN 0097-8418. doi: 10.1145/792548.611966.

[23] National Research Council. *How People Learn: Brain, Mind, Experience, and School: Expanded Edition.* The National Academies Press, Washington, DC, 2000. ISBN 978-0-309-07036-2. doi: 10.17226/9853. URL `https://www.nap.edu/catalog/9853/how-people-learn-brain-mind-experience-and-school-expanded-edition`.

[24] Kathryn Cunningham, Shannon Ke, Mark Guzdial, and Barbara Ericson. Novice rationales for sketching and tracing, and how they try to avoid it. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '19, pages 37–43, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6895-7. doi: 10.1145/3304221.3319788. URL `http://doi.acm.org/10.1145/3304221.3319788`.

[25] Quintin Cutts, Sarah Esper, Marlena Fecho, Stephen R. Foster, and Beth Simon. The abstraction transition taxonomy: Developing desired learning outcomes through the lens of situated cognition. In *Proceedings of the Ninth Annual International Conference on International Computing Education Research*, ICER '12, pages 63–70, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1604-0. doi: 10.1145/2361276.2361290. URL `http://doi.acm.org/10.1145/2361276.2361290`.

[26] Jagannath Das, Jack Naglieri, and John Kirby. *Assessment of cognitive processes: The PASS theory of intelligence.* Allyn & Bacon, 01 1994. ISBN 978-0205141647.

[27] Ian Davidson and Peter Walker. Towards fluid machine intelligence: Can we make a gifted ai? *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:9760–9764, 07 2019. doi: 10.1609/aaai.v33i01.33019760.

[28] R Dawkins. *The Selfish Gene.* Oxford University Press, Oxford, UK, 1976.

[29] Johan de Kleer and John Seely Brown. Assumptions and ambiguities in mechanistic mental models. In *Mental Models*, chapter 8, pages 155–190. Lawrence Erlbaum Associates, Inc., 1983.

[30] Barbara Di Eugenio, Nick Green, Omar AlZoubi, Mehrdad Alizadeh, Rachel Harsley, and Davide Fossati. Worked-out examples in a computer science intelligent tutoring system. In *Proceedings of the 16th Annual Conference on Information Technology Education*, SIGITE 15, page 121, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450338356. doi: 10.1145/2808006.2808011. URL `https://doi.org/10.1145/2808006.2808011`.

[31] Edsger W. Dijkstra. On the cruelty of really teaching computing science. Circulated privately., December 1988. URL `http://www.cs.utexas.edu/users/EWD/ewd10xx/EWD1036.PDF`.

[32] Andrea A. diSessa. Phenomenology and the evolution of intuition. In *Mental Models*, chapter 2. Lawrence Erlbaum Associates, Inc., 1983.

[33] Andrea A diSessa. Knowledge in pieces. In *Constructivism in the Computer Age*. Lawrence Erlbaum Publishers, 1988.

[34] Benedict du Boulay. Some Difficulties of Learning to Program. In E. Soloway and J. C. Spohrer, editors, *Studying the Novice Programmer*, pages 283–299. Lawrence Erlbaum Associates, 1989. URL `http://www.sussex.ac.uk/Users/bend/papers/diffsofprogramming.pdf`.

[35] Benedict du Boulay, Tim O'Shea, and John Monk. The black box inside the glass box: presenting computing concepts to novices. *International Journal of Man-Machine Studies*, 14 (3):237 – 249, 1981. ISSN 0020-7373. doi: https://doi.org/10.1016/S0020-7373(81)80056-9. URL `http://www.sciencedirect.com/science/article/pii/S0020737381800569`.

[36] Reinders Duit, Wolff-Michael Roth, Michael Komorek, and Jens Wilbers. Fostering conceptual change by analogies - between scylla and charybdis. *Learning and Instruction*, 11:283–303, 08 2001. doi: 10.1016/S0959-4752(00)00034-7.

[37] John Duncan, Daphne Chylinski, Daniel Mitchell, and Apoorva Bhandari. Complexity and compositionality in fluid intelligence. *Proceedings of the National Academy of Sciences*, 114: 201621147, 05 2017. doi: 10.1073/pnas.1621147114.

[38] Square Enix. Kingdom hearts 2. Playstation 2, 2005.

[39] Sercan Erer. A comprehensive analysis of students experiences of belonging to the cs community. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE 20, page 579580, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368742. doi: 10.1145/3341525.3394005. URL `https://doi.org/10.1145/3341525.3394005`.

[40] J. Evans and Keith Frankish. *In two minds: Dual processes and beyond.* Oxford University Press, 01 2012. doi: 10.1093/acprof:oso/9780199230167.001.0001.

[41] Brian Falkenhainer, Kenneth D. Forbus, and Dedre Gentner. The structure-mapping engine: Algorithm and examples. *Artif. Intell.*, 41(1):163, November 1989. ISSN 0004-3702. doi: 10.1016/0004-3702(89)90077-5. URL `https://doi.org/10.1016/0004-3702(89)90077-5`.

[42] Sally Fincher and Anthony Robins. *The Cambridge Handbook of Computing Education Research.* Cambridge University Press, 02 2019. doi: 10.1017/9781108654555.002.

[43] Kenneth D. Forbus and Dedre Gentner. Structural evaluation of analogies: What counts? In *Proceedings of CogSci89*, 1989.

[44] Kenneth D. Forbus, Dedre Gentner, and Melissa M. Wu. Towards a computational model of evaluating and using analogical inferences. In *Cognitive Science - COGSCI*, 1997.

[45] Michal Forišek and Monika Steinová. Metaphors and analogies for teaching algorithms. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, pages 15–20, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1098-7. doi: 10.1145/2157136.2157147. URL `http://doi.acm.org/10.1145/2157136.2157147`.

[46] Paulo Freire. *Pedagogy of the Oppressed (50th Anniversary Edition).* Bloosbury Academic, 1970. ISBN 978-1501314131.

[47] Howard Gardner. *Frames of Mind: The Theory of Multiple Intelligences.* Basic Books, 1983.

[48] Geneva Gay. Preparing for culturally responsive teaching. *Journal of Teacher Education*, 53(2):106–116, 2002. doi: 10.1177/0022487102053002003. URL `https://doi.org/10.1177/0022487102053002003`.

[49] Dedre Gentner. Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, 7(2):155 – 170, 1983. ISSN 0364-0213. doi: https://doi.org/10.1016/S0364-0213(83)80009-3. URL `http://www.sciencedirect.com/science/article/pii/S0364021383800093`.

[50] Dedre Gentner and Brian Bowdle. Metaphor as structure-mapping. In Jr. Gibbs and Raymond W, editors, *The Cambridge Handbook of Metaphor and Thought*, pages 109–128. Cambridge University Press, 2008.

[51] Dedre Gentner and Kenneth D. Forbus. Computational models of analogy. *Wiley interdisciplinary reviews. Cognitive science*, 2 3:266–276, 2011.

[52] Dedre Gentner and Donald R. Gentner. Flowing waters or teeming crowds: Mental models of electricity. In Dedre Gentner and Albert L Stevens, editors, *Mental Models*, pages 99–129. Erlbaum, 1983.

[53] Dedre Gentner and Christian Hoyos. Analogy and abstraction. *Topics in Cognitive Science*, 9 (3):672–693, 2017. doi: 10.1111/tops.12278. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/tops.12278`.

[54] Dedre Gentner and Albert L Stevens. *Mental Models*. Erlbaum, 1983.

[55] Dedre Gentner and Cecile Toupin. Systematicity and surface similarity in the development of analogy. *Cognitive Science*, 10(3):277 – 300, 1986. ISSN 0364-0213. doi: https://doi.org/10.1016/S0364-0213(86)80019-2. URL `http://www.sciencedirect.com/science/article/pii/S0364021386800192`.

[56] Dedre Gentner, Brian Bowdle, Phillip Wolff, and Consuelo Boronat. Metaphor is like analogy. In Dedre Gentner, Keith J Holyoak, and Boicho N Kokinov, editors, *The analogical mind: Perspectives from cognitive science*, pages 199–253. MIT Press, 2001.

[57] Dedre Gentner, Jeffrey Loewenstein, and Leigh Thompson. Learning and transfer: A general role for analogical encoding. *JOURNAL OF EDUCATIONAL PSYCHOLOGY*, 95:393–408, 2003.

[58] Dedre Gentner, Jeffrey Loewenstein, Leigh Thompson, and Kenneth D. Forbus. Reviving inert knowledge: Analogical abstraction supports relational retrieval of past events. *Cognitive Science*, 33(8):1343–1382, 2009. doi: 10.1111/j.1551-6709.2009.01070.x. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1551-6709.2009.01070.x`.

[59] Michail Giannakos, Ilias Pappas, Letizia Jaccheri, and Demetrios Sampson. Understanding student retention in computer science education: The role of environment, gains, barriers and usefulness. *Education and Information Technologies*, 10 2016. doi: 10.1007/s10639-016-9538-1.

[60] Mary L. Gick and Keith J. Holyoak. Analogical problem solving. *Cognitive Psychology*, 12(3): 306 – 355, 1980. ISSN 0010-0285. doi: https://doi.org/10.1016/0010-0285(80)90013-4. URL `http://www.sciencedirect.com/science/article/pii/0010028580900134`.

[61] Howard Giles and Tania Ogay. Communication accomodation theory. In *Explaining communication: Contemporary theories and exemplars*, pages 293–310. Lawrence Erlbaum Associates Publishers, 2007. URL `https://psycnet.apa.org/record/2006-21534-016`.

[62] Sam Glucksberg, Patricia Gildea, and Howard B. Bookin. On understanding nonliteral speech: Can people ignore metaphors? In *Journal of Verbal Learning & Verbal Behavior*, pages 85–98, 1982.

[63] Shawn M. Glynn, GA. National Reading Research Center, Athens, and MD. National Reading Research Center, College Park. *Teaching Science with Analogies [microform] : A Strategy for Teachers and Textbook Authors. Reading Research Report No. 15 / Shawn M. Glynn.* Distributed by ERIC Clearinghouse [Washington, D.C.], 1994. URL `https://eric.ed.gov/?id=ED373306`.

[64] Shuchi Grover, Nicholas Jackiw, and Patrik Lundh. Concepts before coding: non-programming interactives to advance learning of introductory programming concepts in middle school. *Computer Science Education*, 29(2-3):106–135, 2019. doi: 10.1080/08993408.2019.1568955. URL `https://doi.org/10.1080/08993408.2019.1568955`.

[65] Marcello Guarini, Amy T. Butchart, Paul L. Simard Smith, and Andrei Moldovan. Resources for research on analogy: A multi-disciplinary guide. *Informal Logic*, 29:84–197, 2009.

[66] Ayush Gupta, David Hammer, and Edward Redish. The case for dynamic models of learners' ontologies in physics. *Journal of the Learning Sciences*, 19, 02 2008. doi: 10.1080/10508406.2010.491751.

[67] Jesper Haglund. Collaborative and self-generated analogies in science education. *Studies in Science Education*, 49:1–34, 03 2013. doi: 10.1080/03057267.2013.801119.

[68] Frank Halasz and Thomas P. Moran. Analogy considered harmful. In *Proceedings of the 1982 Conference on Human Factors in Computing Systems*, CHI '82, pages 383–386, New York, NY, USA, 1982. ACM. doi: 10.1145/800049.801816.

[69] Brian Harrington and Ayaan Chaudhry. Tracademic: Improving participation and engagement in cs1/cs2 with gamified practicals. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE 17, page 347352, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450347044. doi: 10.1145/3059009.3059052. URL `https://doi.org/10.1145/3059009.3059052`.

[70] Rachel Harsley, Nick Green, Mehrdad Alizadeh, Sabita Acharya, Davide Fossati, Barbara Di Eugenio, and Omar AlZoubi. Incorporating analogies and worked out examples as pedagogical strategies in a computer science tutoring system. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 675–680, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-3685-7. doi: 10.1145/2839509.2844637. URL `http://doi.acm.org/10.1145/2839509.2844637`.

[71] Robert R. Hoffman. Monster analogies. *AI Magazine*, 16(3):11, Sep. 1995. doi: 10.1609/aimag.v16i3.1145. URL `https://aaai.org/ojs/index.php/aimagazine/article/view/1145`.

[72] Keith J. Holyoak. *The Spider's Thread: Metaphor in Mind, Brain, and Poetry.* The MIT Press, Feb 2019. ISBN 9780262039222. URL `https://mitpress.mit.edu/books/spiders-thread`.

[73] Keith J. Holyoak and Lindsey E. Richland. Using analogies as a basis for teaching cognitive readiness. In *Teaching and Measuring Cognitive Readiness*, chapter 12. Springer Science and Business Media, 2014. doi: 10.1007/978-1-4614-7579-8_12.

[74] Keith J. Holyoak and Paul Thagard. Analogical mapping by constraint satisfaction. *Cognitive Science*, 13(3):295–355, 1989. doi: 10.1207/s15516709cog1303_1. URL `https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1303_1`.

[75] Grace Hopper. Navy computer grandmother keeps moving (kni). *The Baltimore Sun*, September 1975.

[76] John L. Horn and Raymond B. Cattell. Refinement and test of the theory of fluid and crystallized general intelligences. *Journal of Educational Psychology*, 57(5):253–270, 1966. doi: https://doi.org/10.1037/h0023816.

[77] John E. Hummel, John Licato, and Selmer Bringsjord. Analogy, explanation, and proof. *Frontiers in Human Neuroscience*, 8:867, 2014. ISSN 1662-5161. doi: 10.3389/fnhum.2014.00867. URL https://www.frontiersin.org/article/10.3389/fnhum.2014.00867.

[78] Edwin Hutchins. Understanding micronesian navigation. In Dedre Gentner and Albert Stevens, editors, *Mental Models*, chapter 9, pages 191–225. Lawrence Erlbaum Associates, Inc., 1983.

[79] David James. The use of djing tasks as a pedagogical bridge to learning data structures. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE 20, page 193197, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368742. doi: 10.1145/3341525.3387427. URL https://doi.org/10.1145/3341525.3387427.

[80] David H. Jonassen. Objectivism versus constructivism: Do we need a new philosophical paradigm? *Educational Technology Research and Development*, 39:5–14, 1991.

[81] Chris Jones and Liliane Esnault. The metaphor of networks in learning: Communities, collaboration and practice. In *Networked Learning 2004: a Research Based Conference on E-Learning in Higher Education and Lifelong Learning: Proceedings of the Fourth International Conference on Networked Learning*, 01 2004.

[82] Rex Jung and Richard Haier. The parieto-frontal integration theory (p-fit) of intelligence: Converging neuroimaging evidence. *The Behavioral and brain sciences*, 30:135–54; discussion 154, 05 2007. doi: 10.1017/S0140525X07001185.

[83] Scott Barry Kaufman. Beyond general intelligence: The dual-process theory of human intelligence (doctoral dissertation), 2009.

[84] Scott Barry Kaufman, Colin G. DeYoung, Jeremy R. Gray, Jamie Brown, and Nicholas Mackintosh. Associative learning predicts intelligence above and beyond working memory and processing speed. *Intelligence*, 37(4):374 – 382, 2009. ISSN 0160-2896. doi: https://doi.org/10.1016/j.intell.2009.03.004. URL http://www.sciencedirect.com/science/article/pii/S0160289609000300.

[85] Eunyoung Kim and Razvan Beuran. On designing a cybersecurity educational program for higher education. In *Proceedings of the 10th International Conference on Education Technology and Computers*, ICETC 18, page 195200, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450365178. doi: 10.1145/3290511.3290524. URL https://doi.org/10.1145/3290511.3290524.

[86] Adam Easton Kirk. Memes and metaphors (masters thesis piece), May 2019. URL https://conservancy.umn.edu/bitstream/handle/11299/202944/Memes\%20and\%20Metaphors\%20-\%20Adam\%20Kirk\%20-\%202019\%20-.pdf.

[87] Shriram Krishnamurthi and Kathi Fisler. Programming paradigms and beyond. In *The Cambridge Handbook of Computing Education Research*, 2019. URL https://cs.brown.edu/~sk/Publications/Papers/Published/kf-prog-paradigms-and-beyond.

[88] George Lakoff. The contemporary theory of metaphor. In AndrewEditor Ortony, editor, *Metaphor and Thought*, page 202251. Cambridge University Press, 2 edition, 1993. doi: 10.1017/CBO9781139173865.013.

[89] Jean Lave, Michael Murtaugh, and Olivia de la Rocha. The dialectic of arithmetic in grocery shopping. In *Everyday cognition: Its development in social context*, pages 67–94. Harvard University Press, 1984.

[90] Lucas Layman. Changing students' perceptions: An analysis of the supplementary benefits of collaborative software development. In *Proceedings of the 19th Conference on Software Engineering Education & Training*, CSEET '06, pages 159–166, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2557-1. doi: 10.1109/CSEET.2006.10. URL `https://doi.org/10.1109/CSEET.2006.10`.

[91] Michael J. Lee and Amy J. Ko. Personifying programming tool feedback improves novice programmers learning. In *Proceedings of the Seventh International Workshop on Computing Education Research*, ICER 11, page 109116, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450308298. doi: 10.1145/2016911.2016934.

[92] Cedric J. Linder. A challenge to conceptual change. *Science Education*, 77(3):293–300, 1993. doi: 10.1002/sce.3730770304. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/sce.3730770304`.

[93] Julie Linsey, Arthur Markman, and Kristin Wood. Wordtrees: A method for design-by-analogy. In *American Society for Engineering Education Annual Conference*, 06 2008.

[94] Regina Lipkens and Steven Hayes. Producing and recognizing analogical relations. *Journal of the experimental analysis of behavior*, 91:105–26, 02 2009. doi: 10.1901/jeab.2009.91-105.

[95] Jeffrey Loewenstein. Structure mapping and vocabularies for thinking. *Topics in Cognitive Science*, 9(3):842–858, 2017. doi: 10.1111/tops.12280. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/tops.12280`.

[96] Marsha Lovett, Lynne Reder, and Christian Lebiere. Modeling working memory in a unified architecture: An act-r perspective. *Models of Working Memory: Mechanisms of Active Maintenance and Executive Control*, 01 1999.

[97] Tony Lowe. Explaining novice programmer's struggles, in two parts: Revisiting the iticse 2004 working group's study using dual process theory. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '19, pages 30–36, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6895-7. doi: 10.1145/3304221.3319775. URL `http://doi.acm.org/10.1145/3304221.3319775`.

[98] Linxiao Ma, John Ferguson, Marc Roper, Isla Ross, and Murray Wood. Improving the mental models held by novice programmers using cognitive conflict and jeliot visualisations. In *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE '09, pages 166–170, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-381-5. doi: 10.1145/1562877.1562931. URL `http://doi.acm.org/10.1145/1562877.1562931`.

[99] B. Marín, J. Frez, J. Cruz-Lemus, and M. Genero. An empirical investigation on the benefits of gamification in programming courses. *ACM Trans. Comput. Educ.*, 19(1), November 2018. doi: 10.1145/3231709. URL `https://doi.org/10.1145/3231709`.

[100] James H. Martin. Understanding new metaphors. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'87, pages 137–139, San Francisco, CA,

USA, 1987. Morgan Kaufmann Publishers Inc. URL `http://dl.acm.org/citation.cfm?id=1625015.1625041`.

[101] John D. Mayer, Peter Salovey, and David R. Caruso. Mayer-salovey-caruso emotional intelligence test (msceit) users manual, 2002. URL `https://storefront.mhs.com/collections/msceit`.

[102] Sarah J. McCarthey and Elizabeth Birr Moje. Identity matters. *Reading Research Quarterly*, 37(2):228–238, 2002. doi: 10.1598/RRQ.37.2.6. URL `https://ila.onlinelibrary.wiley.com/doi/abs/10.1598/RRQ.37.2.6`.

[103] Michael McCloskey. Naive theories of motion. In Dedre Gentner and Albert Stevens, editors, *Mental Models*, chapter 13, pages 299–322. Lawrence Erlbaum Associates, Inc., 1983.

[104] Roger McDermott, Gordon Eccleston, and Garry Brindley. More than a good story can you really teach programming through storytelling? *Innovation in Teaching and Learning in Information and Computer Sciences*, 7(1):34–43, 2008. doi: 10.11120/ital.2008.07010034. URL `https://doi.org/10.11120/ital.2008.07010034`.

[105] Know Your Meme. It's free real estate, 2009. URL `https://knowyourmeme.com/memes/its-free-real-estate`. Image from "Tim & Eric Awesome Show, Great Job!" (2009).

[106] Know Your Meme. Who would win?, nov 2010. URL `https://knowyourmeme.com/memes/who-would-win`.

[107] Know Your Meme. "is this a pigeon?", dec 2011. URL `https://knowyourmeme.com/memes/is-this-a-pigeon`. Image from "The Brave Fighter of Sun Fighbird" (1991).

[108] Know Your Meme. Does he bite?, nov 2015. URL `https://knowyourmeme.com/memes/does-he-bite`. Image from "Cyanide and Happiness" (2015).

[109] Know Your Meme. "whatcha got there?", nov 2015. URL `https://knowyourmeme.com/memes/whatcha-got-there`. Image from "iCarly" episode "iMeet Fred" (2009).

[110] Know Your Meme. "roll safe", nov 2016. URL `https://knowyourmeme.com/memes/roll-safe`. Image from "Hood Documentary" series (2016).

[111] Know Your Meme. Gru's plan, mar 2018. URL `https://knowyourmeme.com/memes/grus-plan`. Image from "Despicable Me" (2010).

[112] Know Your Meme. Ight imma head out, jul 2019. URL `https://knowyourmeme.com/memes/ight-imma-head-out`. Image from "Spongebob Squarepants" episode "The Smoking Peanut" (2001).

[113] George Abram Miller. The magical number seven plus or minus two: some limits on our capacity for processing information. *Psychological review*, 63 2:81–97, 1956.

[114] Ryan M Milner. *The world made meme: public conversations and participatory media.* The MIT Press, 2016. doi: https://doi.org/10.7551/mitpress/9780262034999.001.0001.

[115] Catherine Mooney, Anna Antoniadi, Ioannis Karvelas, Lána Salmon, and Brett A. Becker. Exploring sense of belonging in computer science students. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE 20, page 563, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368742. doi: 10.1145/3341525.3393974. URL `https://doi.org/10.1145/3341525.3393974`.

[116] Alberto Mora, Elena Planas, and Joan Arnedo-Moreno. Designing game-like activities to engage adult learners in higher education. In *Proceedings of the Fourth International Conference on Technological Ecosystems for Enhancing Multiculturality*, TEEM 16, page 755762, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450347471. doi: 10.1145/3012430.3012603. URL `https://doi.org/10.1145/3012430.3012603`.

[117] Briana Morrison, Lauren Margulieux, and Adrienne Decker. Using subgoal labeling in teaching cs1. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE 19, page 1237, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450358903. doi: 10.1145/3287324.3287540. URL `https://doi.org/10.1145/3287324.3287540`.

[118] Peter Morville. User experience design, 2004. URL `http://semanticstudios.com/user_experience_design/`.

[119] Orna Muller, David Ginat, and Bruria Haberman. Pattern-oriented instruction and its influence on problem decomposition and solution construction. In *Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE 07, page 151155, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595936103. doi: 10.1145/1268784.1268830. URL `https://doi.org/10.1145/1268784.1268830`.

[120] Nancy Nersessian. *Creating Scientific Concepts*. The MIT Press, 01 2008. ISBN 9780262280549. doi: 10.7551/mitpress/7967.001.0001.

[121] Mathilde Neugnot-Cerioli, Charlotte Gagner, and Miriam H. Beauchamp. Training of fluid and crystallized intelligence: A game-based approach in adolescents presenting with below average iq. *Cogent Psychology*, 4(1):1284360, 2017. doi: 10.1080/23311908.2017.1284360. URL `https://www.tandfonline.com/doi/abs/10.1080/23311908.2017.1284360`.

[122] Asaf Nissenbaum and Limor Shifman. Meme templates as expressive repertoires in a globalizing world: A cross-linguistic study. *Journal of Computer-Mediated Communication*, 23(5):294–310, 08 2018. ISSN 1083-6101. doi: 10.1093/jcmc/zmy016. URL `https://doi.org/10.1093/jcmc/zmy016`.

[123] Safiya Umoja Noble. *Algorithms of Oppression: How Search Engines Reinforce Racism*. NYU Press, 2018. ISBN 9781479849949. URL `http://www.jstor.org/stable/j.ctt1pwt9w5`.

[124] Don Norman and Jakob Nielsen. The definition of user experience (ux), 2020. URL `https://www.nngroup.com/articles/definition-user-experience/`.

[125] Donald A. Norman. Some observations on mental models. In Dedre Gentner and Albert Stevens, editors, *Mental Models*, chapter 1, pages 7–14. Lawrence Erlbaum Associates, Inc., 1983.

[126] Donald A. Norman. *The Design of Everyday Things*. Basic Books, Inc., New York, NY, USA, 2002. ISBN 9780465067107.

[127] Benjamin D Nye. Modeling memes: A memetic view of affordance learning (dissertation), Spring 2011. URL `https://repository.upenn.edu/edissertations/336`.

[128] Linda Ott, Briana Bettin, and Leo Ureel. The impact of placement in introductory computer science courses on student persistence in a computing major. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE 2018, page 296301, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450357074. doi: 10.1145/3197091.3197139. URL `https://doi.org/10.1145/3197091.3197139`.

[129] David Perkins. Deep and fragile knowledge. In *Smart Schools*. Free Press, 2003. URL `https://learner.org/wp-content/uploads/2019/02/1.DeepFragileKnowledge.pdf`.

[130] David N. Perkins and Fay Martin. Fragile knowledge and neglected strategies in novice programmers. *Educational Technology Center*, 1985.

[131] Franziska Preusse, Elke van der Meer, Gopikrishna Deshpande, Frank Krueger, and Isabell Wartenburger. Fluid intelligence allows flexible recruitment of the parieto-frontal network in analogical reasoning. *Frontiers in human neuroscience*, 5:22, 03 2011. doi: 10.3389/fnhum.2011.00022.

[132] Yizhou Qian and James Lehman. Students misconceptions and other difficulties in introductory programming: A literature review. *ACM Trans. Comput. Educ.*, 18(1), October 2017. doi: 10.1145/3077618. URL `https://doi.org/10.1145/3077618`.

[133] Daniel Reisberg, James C. Kaufman, Scott Barry Kaufman, and Jonathan A. Plucker. Contemporary theories of intelligence, 06 2013. URL `https://www.oxfordhandbooks.com/view/10.1093/oxfordhb/9780195376746.001.0001/oxfordhb-9780195376746-e-51`.

[134] World Population Review. Houghton, michigan population 2020, 2020. URL `https://worldpopulationreview.com/us-cities/houghton-mi-population`.

[135] World Population Review. Marion, wisconsin population 2020, 2020. URL `https://worldpopulationreview.com/us-cities/marion-wi-population`.

[136] Jean J. Ryoo, Cynthia Estrada, Tiera Tanksley, and Jane Margolis. Connecting computer science education to students passions: A critical step toward supporting equity in cs education. *UCLA Center X*, 2019. URL `https://centerx.gseis.ucla.edu/wp-content/uploads/2019/05/Student-Passions_Interests-in-CS-Learning-Contexts-Working-Paper_03192019.pdf`.

[137] Joseph P. Sanford, Aaron Tietz, Saad Farooq, Samuel Guyer, and R. Benjamin Shapiro. Metaphors we teach by. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 585–590, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2605-6. doi: 10.1145/2538862.2538945.

[138] Ching-Kuang Shene. Threadmentor, the dining philosophers problem, 2020. URL `https://pages.mtu.edu/~shene/NSF-3/e-Book/MUTEX/TM-example-philos-1.html`.

[139] Chaklam Silpasuwanchai, Xiaojuan Ma, Hiroaki Shigemasu, and Xiangshi Ren. Developing a comprehensive engagement framework of gamification for reflective learning. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems*, DIS 16, page 459472, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450340311. doi: 10.1145/2901790.2901836. URL `https://doi.org/10.1145/2901790.2901836`.

[140] Ben Skudder and Andrew Luxton-Reilly. Worked examples in computer science. In *Proceedings of the Sixteenth Australasian Computing Education Conference - Volume 148*, ACE 14, page 5964, AUS, 2014. Australian Computer Society, Inc. ISBN 9781921770319.

[141] James D. Slotta and Michelene T. H. Chi. Helping students understand challenging topics in science through ontology training. *Cognition and Instruction*, 24:261 – 289, 2006.

[142] Amber Solomon, Mark Guzdial, Betsy DiSalvo, and Ben Rydal Shapiro. Applying a gesture taxonomy to introductory computing concepts. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*, ICER 18, page 250257, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356282. doi: 10.1145/3230977.3231001. URL `https://doi.org/10.1145/3230977.3231001`.

[143] Juha Sorva. Notional machines and introductory programming education. *Trans. Comput. Educ.*, 13(2):8:1–8:31, July 2013. ISSN 1946-6226. doi: 10.1145/2483710.2483713. URL `http://doi.acm.org/10.1145/2483710.2483713`.

[144] Lesley Spier-Dance, Jolie Mayer-Smith, Nigel Dance, and Samia Khan. The role of student-generated analogies in promoting conceptual understanding for undergraduate chemistry students. *Research in Science and Technological Education*, 23, 02 2005. doi: 10.1080/02635140500266401.

[145] Rand Spiro, Paul J. Feltovich, Richard Coulson, and Daniel Anderson. Multiple analogies for complex concepts : antidotes for analogy-induced misconception in advanced knowledge acquisition. In *Similarity and analogical reasoning*. Cambridge University Press, 10 1987.

[146] Andreas Stefik and Susanna Siebert. An empirical investigation into programming language syntax. *ACM Trans. Comput. Educ.*, 13(4), November 2013. doi: 10.1145/2534973. URL `https://doi.org/10.1145/2534973`.

[147] Robert J. Sternberg. *Successful Intelligence: How Practical and Creative Intelligence Determine Success in Life*. Plume, October 1997. ISBN 978-0452279063.

[148] Sangho Suh. Using concreteness fading to model &#38; design learning process. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*, ICER '19, pages 353–354, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6185-9. doi: 10.1145/3291279.3339445. URL `http://doi.acm.org/10.1145/3291279.3339445`.

[149] Keith Taber. *Alternative Conceptions/Frameworks/Misconceptions*, pages 1–5. Springer, 01 2014. ISBN 978-94-007-6165-0. doi: 10.1007/978-94-007-6165-0_88-2.

[150] Elli J. Theobald, Sarah L. Eddy, Daniel Z. Grunspan, Benjamin L. Wiggins, and Alison J. Crowe. Student perception of group dynamics predicts individual performance: Comfort and equity matter. *PLOS ONE*, 12(7):1–16, 07 2017. doi: 10.1371/journal.pone.0181336. URL `https://doi.org/10.1371/journal.pone.0181336`.

[151] Lynda Thomas, Mark Ratcliffe, and Ann Robertson. Code warriors and code-a-phobes: A study in attitude and pair programming. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '03, pages 363–367, New York, NY, USA, 2003. ACM. ISBN 1-58113-648-X. doi: 10.1145/611892.612007. URL `http://doi.acm.org/10.1145/611892.612007`.

[152] Vincent Tinto. Classrooms as communities: Exploring the educational character of student persistence. *The Journal of Higher Education*, 68(6):599–623, 1997. ISSN 00221546, 15384640. URL `http://www.jstor.org/stable/2959965`.

[153] CS Unplugged. Cs unplugged: Computer science without a computer, 2020. URL `https://csunplugged.org/en/`.

[154] Leo C. Ureel II and Charles Wallace. Automated critique of early programming antipatterns. SIGCSE '19, page 738744, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450358903. doi: 10.1145/3287324.3287463. URL `https://doi.org/10.1145/3287324.3287463`.

[155] Usability.Gov. User experience basics, 2020. URL `https://www.usability.gov/what-and-why/user-experience.html`.

[156] Vesa Vainio and Jorma Sajaniemi. Factors in novice programmers poor tracing skills. In *Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE 07, page 236240, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595936103. doi: 10.1145/1268784.1268853. URL `https://doi.org/10.1145/1268784.1268853`.

[157] Han L J van der Maas, Conor V Dolan, Raoul P P P Grasman, Jelte M Wicherts, Hilde M Huizenga, and Maartje E J Raijmakers. A dynamical model of general intelligence: the positive manifold of intelligence by mutualism. *Psychological Review*, 4, October 2006. doi: 10.1037/0033-295X.113.4.842.

[158] Nanette Veilleux, Rebecca Bates, Cheryl Allendoerfer, Diane Jones, Joyous Crawford, and Tamara Floyd Smith. The relationship between belonging and ability in computer science. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE 13, page 6570, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450318686. doi: 10.1145/2445196.2445220. URL `https://doi.org/10.1145/2445196.2445220`.

[159] Melissa B. Wanzer, Ann B. Frymier, and Jeffrey Irwin. An explanation of the relationship between instructor humor and student learning: Instructional humor processing theory. *Communication Education*, 59(1):1–18, 2010. doi: 10.1080/03634520903367238. URL `https://doi.org/10.1080/03634520903367238`.

[160] Alicia Nicki Washington. When twice as good isnt enough: The case for cultural competence in computing. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, SIGCSE 20, page 213219, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450367936. doi: 10.1145/3328778.3366792. URL `https://doi.org/10.1145/3328778.3366792`.

[161] Etienne Wenger. *Communities of Practice: Learning, Meaning, and Identity.* Learning in Doing: Social, Cognitive and Computational Perspectives. Cambridge University Press, 1998. doi: 10.1017/CBO9780511803932.

[162] Avi Wigderson. Knowledge, creativity and p versus np, 2009.

[163] Wikipedia. Limerick (poetry), July 2020. URL `https://en.wikipedia.org/wiki/Limerick_(poetry)`.

[164] Michael D Williams, James D Hollan, and Albert L Stevens. Human reasoning about a simple physical system. In *Mental Models*, chapter 7, pages 131–153. Lawrence Erlbaum Associates, Inc., 1983.

[165] L. a. Wozny. The application of metaphor, analogy, and conceptual models in computer systems. *Interact. Comput.*, 1(3):273–283, December 1989. ISSN 0953-5438. doi: 10.1016/0953-5438(89)90015-5. URL `https://doi.org/10.1016/0953-5438(89)90015-5`.

[166] Randy Yerrick, Elizabeth Doster, Jeff Nugent, Helen Parke, and Frank Crawley. Social interaction and the use of analogy: An analysis of preservice teachers' talk during physics inquiry lessons. *Journal of Research in Science Teaching*, 40:443 – 463, 05 2003. doi: 10.1002/tea.10084.

[167] Richard M Young. Surrogates and mappings: Two kinds of conceptual models for interactive devices. In *Mental Models*, chapter 3, pages 35–52. Lawrence Erlbaum Associates, Inc., 1983.

[168] Kun Yuan, Jeffrey Steedle, Richard Shavelson, Alicia Alonzo, and Marily Oppezzo. Working memory, fluid intelligence, and science learning. *Educational Research Review*, 1(2):83 – 98, 2006. ISSN 1747-938X. doi: https://doi.org/10.1016/j.edurev.2006.08.005. URL `http://www.sciencedirect.com/science/article/pii/S1747938X06000285`.

[169] Hassan Hussein Zeitoun. Teaching scientific analogies: a proposed model. *Research in Science & Technological Education*, 2(2):107–125, 1984. doi: 10.1080/0263514840020203. URL `https://doi.org/10.1080/0263514840020203`.

[170] Kevin B. Zook. Effects of analogical processes on learning and misrepresentation. *Educational Psychology Review*, 3(1):41–72, 1991. ISSN 1040726X, 1573336X. URL `http://www.jstor.org/stable/23359215`.

Appendices

Appendix A

CS1 OPAL Initial Analogies

Table A.1: Variable Names are Case Sensitive - Passwords

Identification of Analogy Context	
Misconception	Capitalization of variable does not matter
Desired Knowledge	Variable names are case sensitive
Exploration of Target Domain (Programming) Procedure	
Precondition	A previously declared variable name.
Required Action	The variable name must be replicated exactly.
Postcondition	Ability to use the value of the variable.
Constraints	Upper and lowercase versions of the same letter are distinct.
Exploration of Source Domain Procedure	
Domain	Passwords
Precondition	A previously declared password.
Required Action	The password must be replicated exactly.
Postcondition	Ability to access password protected information.
Constraints	Different gestures, character representations, vocalizations, or attribute presentations are distinct.
Analysis of Common Structural Elements	
Precondition	A previously created key.
Required Action	An exact replication of the defining key details.
Postcondition	Access to whatever the key is mapped to.
Constraints	Any alternate representations of defining key components are distinct.

Table A.2: Variable Types - Introductions

Identification of Analogy Context	
Misconception	Type restated on variable reuse
Desired Knowledge	Declaration of type is associated with variable name
Exploration of Target Domain (Programming) Procedure	
Precondition	An already initialized variable.
Required Action	Using only the variable's name to reference it.
Postcondition	Access to the information the variable stores.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Introductions
Precondition	Someone is introduced with their name, job, and age.
Required Action	Referring to the person by name.
Postcondition	The person knows you are speaking to them.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	Information associated with a previously defined name and specification.
Required Action	Restating the name of the information.
Postcondition	Access to the information associated with the name.
Constraints	None.

Table A.3: Scanner Storage - Mail Sorting

Identification of Analogy Context	
Misconception	Scanners store inputs to variables automatically
Desired Knowledge	Obtained values must be stored immediately or they are lost
Exploration of Target Domain (Programming) Procedure	
Precondition	An initialized Scanner.
Required Action	Call next methods to obtain input.
Postcondition	Scanner has current input at time of next call.
Constraints	Information is only held at time of next call, and discarded after.
Exploration of Source Domain Procedure	
Domain	Sorting Mail
Precondition	A mailbox with mail in it.
Required Action	Visit mailbox to get mail.
Postcondition	You have the mail in your hand after visiting box.
Constraints	Without storing the mail somewhere, you are likely to lose it.
Analysis of Common Structural Elements	
Precondition	A means of obtaining input information.
Required Action	Obtain information one piece at a time.
Postcondition	Current piece of information is shown when obtained.
Constraints	Information shown is lost unless stored.

Table A.4: Multiple Scanners - Office Printers

Identification of Analogy Context	
Misconception	Multiple scanner creation for single input source
Desired Knowledge	Only one scanner per location is necessary
Exploration of Target Domain (Programming) Procedure	
Precondition	An initialized Scanner.
Required Action	Invoke next methods on the same scanner to obtain input.
Postcondition	Scanner will obtain further input.
Constraints	Additional Scanners initialized to the same location will cause issues.
Exploration of Source Domain Procedure	
Domain	Office Printers
Precondition	A copy machine in an office workroom.
Required Action	Send items to print to machine remotely using commands.
Postcondition	The machine obtains commands and prints jobs.
Constraints	Additional copy machines in the same workroom may cause confusion as to which machine has your printout.
Analysis of Common Structural Elements	
Precondition	A means of obtaining input from a location.
Required Action	Continuing to utilize the same input means.
Postcondition	Obtain further input from that location.
Constraints	Multiple obtainers at same location causes confusion.

Table A.5: Reserved Words - Crowded Space

Identification of Analogy Context	
Misconception	Variables can be named reserved words
Desired Knowledge	Reserved words have special meaning in Java
Exploration of Target Domain (Programming) Procedure	
Precondition	A reserved word.
Required Action	Typing the word in Java code.
Postcondition	Reserved word invokes its meaning in Java.
Constraints	Can only convey its Java meaning.
Exploration of Source Domain Procedure	
Domain	Crowded Space
Precondition	Person existing in a crowded space.
Required Action	Yelling Fire.
Postcondition	Public assumes a fire is occurring.
Constraints	Fire conveys one meaning in a crowded environment.
Analysis of Common Structural Elements	
Precondition	Information with special meaning.
Required Action	Use the information.
Postcondition	Meaning of the information is conveyed.
Constraints	Can only convey one meaning.

Table A.6: Primitive Wrappers - Ordering Fast Food

Identification of Analogy Context	
Misconception	Primitives and their wrapper classes are identical
Desired Knowledge	Primitives and reference types are distinct
Exploration of Target Domain (Programming) Procedure	
Precondition	A situation where a primitive that has a wrapper class is needed.
Required Action	If only value is needed, primitive is used.
Postcondition	Value is accessed simply.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Ordering Fast Food.
Precondition	A burger and a burger wrapper.
Required Action	If eating the burger immediately, wrapper is unnecessary.
Postcondition	Burger can be accessed without wrapper overhead.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	Element with value and element representing element with value.
Required Action	When only value is needed element with value is used.
Postcondition	Simplest access to value is carried out.
Constraints	None.

Table A.7: ASCII Shift - Moving to a New Place

Identification of Analogy Context	
Misconception	ASCII bit shift does not affect non-alphabetic characters
Desired Knowledge	Shift affects any ASCII characters unless a condition states otherwise
Exploration of Target Domain (Programming) Procedure	
Precondition	Characters of text containing ones to shift to uppercase.
Required Action	Must check if character is alphabetical and lowercase before shifting.
Postcondition	Only lowercase characters change, others stay same.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Moving to a new place
Precondition	A form with information fields, including address.
Required Action	Check if the area of the form is one that should change with the move.
Postcondition	Only the address information changes, all other details stay the same.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	Information that should change values.
Required Action	Only specific types of information in the set should change values.
Postcondition	Appropriately modified information with other values staying same.
Constraints	None.

Table A.8: Scanner Delimiters - Surveys

Identification of Analogy Context	
Misconception	Scanner requires a delimiter to be set up
Desired Knowledge	Scanner has a default delimiter on instantiation
Exploration of Target Domain (Programming) Procedure	
Precondition	A Scanner.
Required Action	Call next methods to get information.
Postcondition	Information is obtained.
Constraints	Creating the Scanner defines a delimiter.
Exploration of Source Domain Procedure	
Domain	Surveys.
Precondition	An administered survey's responses.
Required Action	Clicking "next" or flipping pages moves to next survey.
Postcondition	Information from the viewed survey can be obtained.
Constraints	When the survey was created it was given an end point which you use to distinguish different surveys.
Analysis of Common Structural Elements	
Precondition	A means of obtaining input information.
Required Action	Read in input information after setup.
Postcondition	Input information is gathered.
Constraints	Setup defines a way to distinguish input separations.

Table A.9: Division and Floating Points - Cooking

Identification of Analogy Context	
Misconception	Setting the result to double will stop integer division truncation
Desired Knowledge	Floating point must be introduced to the division for floating point results to be remembered
Exploration of Target Domain (Programming) Procedure	
Precondition	Two integers being divided.
Required Action	Division introduces a double.
Postcondition	The result can be a double with fractional elements retained.
Constraints	Double must occur during division not only as the result.
Exploration of Source Domain Procedure	
Domain	Cooking.
Precondition	A recipe containing wheat.
Required Action	Gluten allergies require ingredient inclusion during steps.
Postcondition	The result modifies the recipe to be gluten-free.
Constraints	Ingredients need to be changed during recipe preparation not after.
Analysis of Common Structural Elements	
Precondition	Action utilizing one type of information.
Required Action	A different type of information is introduced to the steps of operation.
Postcondition	The different information can be assimilated into the result.
Constraints	Information must be introduced during operational steps, not after.

Table A.10: Primitives Copy Values - Writing Notes

Identification of Analogy Context	
Misconception	Primitive assignments maintain references
Desired Knowledge	Primitive assignments copy a value to another primitive
Exploration of Target Domain (Programming) Procedure	
Precondition	Two primitive variables.
Required Action	One primitive is assigned the value of another primitive.
Postcondition	Both have a distinct copy of that value.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Writing Notes.
Precondition	Yours and a classmate's notebooks.
Required Action	You use your classmate's notebook to copy over notes you missed.
Postcondition	Both of you have distinct copies of the notes.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	Two elements holding information.
Required Action	One element copies the information of another element.
Postcondition	Both elements have distinct copies of the information.
Constraints	None.

Table A.11: Class Contains Code - Cinema

Identification of Analogy Context	
Misconception	Code can be anywhere in a java file
Desired Knowledge	The code should be contained within a class
Exploration of Target Domain (Programming) Procedure	
Precondition	A java class.
Required Action	Code is contained within the java class.
Postcondition	The code is part of the class and can be executed.
Constraints	Code outside a class it not runnable.
Exploration of Source Domain Procedure	
Domain	Cinema.
Precondition	A play or movie.
Required Action	Details of the plot are inside the runtime.
Postcondition	The movie or play covers the details of its plot.
Constraints	Anything beyond the runtime is not included in the movie and not seen.
Analysis of Common Structural Elements	
Precondition	A specification.
Required Action	Details of the specification are inside it.
Postcondition	The specification contains its information.
Constraints	Information outside the specification is not included in it.

Table A.12: Semicolons End - Reading

Identification of Analogy Context	
Misconception	Semicolons are not needed to end statements
Desired Knowledge	Semicolons should end all regular statements
Exploration of Target Domain (Programming) Procedure	
Precondition	A statement.
Required Action	A semicolon to end the statement.
Postcondition	The information in the statement can be analyzed and understood.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Reading.
Precondition	Several sentences expressing an idea.
Required Action	A period ends each sentence.
Postcondition	We can read and understand each part of the idea in turn.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	An action or expression of information.
Required Action	A way of knowing when the statement is completed.
Postcondition	The action can be understood and/or executed.
Constraints	None.

Table A.13: Operations, then Assignment - Savings

Identification of Analogy Context	
Misconception	A variable can't reference itself during assignment
Desired Knowledge	Operations occur before an assignment
Exploration of Target Domain (Programming) Procedure	
Precondition	A variable.
Required Action	The value is changed using the previous value.
Postcondition	The old value is used then lost when the assignment completes.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Savings.
Precondition	A bank account.
Required Action	Your balance changes based on the previous value.
Postcondition	The old value is used and then lost when your balance updates to the current value.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A named element with a value.
Required Action	Changing the value of the element using its previous value.
Postcondition	The value can be used during the operation to obtain a new value.
Constraints	None.

Table A.14: Return Ends Method - Exploration

Identification of Analogy Context	
Misconception	Methods keep executing after a return statement
Desired Knowledge	The method immediately ceases execution and returns any value stated in the return statement
Exploration of Target Domain (Programming) Procedure	
Precondition	A method that is at a return statement.
Required Action	The method ends immediately and the return value is sent to caller.
Postcondition	The method completes any steps after return do not execute.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Exploration.
Precondition	A mine expedition yielded searched-for treasure.
Required Action	The expedition ends and the treasure is sent back.
Postcondition	The crew does not stick around - they found what they were looking for.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A set of actions that has generated a result.
Required Action	The actions end immediately and the result is sent back.
Postcondition	The action has completed and does not do any other steps that it may have.
Constraints	None.

Table A.15: Independent Scopes - People Names

Identification of Analogy Context	
Misconception	Variables with the same name in different scopes have the same values
Desired Knowledge	Separate scopes are independent
Exploration of Target Domain (Programming) Procedure	
Precondition	Two variables with the same name in distinct scopes.
Required Action	The variables act independently.
Postcondition	The variables have independent values of each other.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	People Names.
Precondition	Two people with the same name in different locations.
Required Action	Lives are independent of each other.
Postcondition	Do not know details of each other and act as such.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	Two things with the same name in isolated locations.
Required Action	The things operate independent of each other.
Postcondition	The things do not know details of each other and act as such.
Constraints	None.

Table A.16: Cast Affects Immediate Next Element - Dye Work

Identification of Analogy Context	
Misconception	Casting occurs after all operations in an expression
Desired Knowledge	Casting will immediately cast the element after it with only parentheses having higher precedent
Exploration of Target Domain (Programming) Procedure	
Precondition	A cast.
Required Action	Cast is applied to the next element.
Postcondition	The next element is changed to the cast type.
Constraints	Parentheses can be used to combine multiple operations before the cast.
Exploration of Source Domain Procedure	
Domain	Dye Work.
Precondition	Bottle of dye.
Required Action	The dye is applied to a single container.
Postcondition	The contents of the container are affected by the dye but other containers are not.
Constraints	Multiple containers contents can be added to a single container to have all of those contents affected by the dye.
Analysis of Common Structural Elements	
Precondition	A type changer for elements.
Required Action	Type change happens to the contained information directly next to it.
Postcondition	The type change is applied to the information.
Constraints	Actions can increase how much information the direct next container has.

Table A.17: Methods are Not Defined Within Methods - Using a Computer

Identification of Analogy Context	
Misconception	Methods can be defined in other methods
Desired Knowledge	Methods can call other methods but each method should be defined distinctly
Exploration of Target Domain (Programming) Procedure	
Precondition	A method.
Required Action	A method which calls another method.
Postcondition	The other method is jumped to finishes and returns to the previous method.
Constraints	The other method is not defined in the first method.
Exploration of Source Domain Procedure	
Domain	Using a Computer.
Precondition	A computer program you want to run.
Required Action	Steps may tell you to download or use another program.
Postcondition	You follow the step and continue.
Constraints	The downloaded program was required but was not included in your program's process.
Analysis of Common Structural Elements	
Precondition	A process containing several steps.
Required Action	The steps refer to another process to complete.
Postcondition	That process is referred to and the steps continue.
Constraints	The other process is not defined inside the first process.

Table A.18: Header Comment Importance - Pet Care

Identification of Analogy Context	
Misconception	Header comments are not important
Desired Knowledge	Header comments are vital to other programmers attempting to use your code
Exploration of Target Domain (Programming) Procedure	
Precondition	A method.
Required Action	Adding header comments to explain the method.
Postcondition	Users of the method understand how and why to use it.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Pet Care.
Precondition	Someone is taking care of your pet.
Required Action	Details for routines allergies needs and more.
Postcondition	The caretaker can appropriately care for your pet and understand needed actions.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A series of actions.
Required Action	Instructions for why these actions are performed.
Postcondition	Someone else can understand the use of performing the actions.
Constraints	None.

Table A.19: Arguments Send Information - Taking an Order

Identification of Analogy Context	
Misconception	Arguments ignored with Scanner input
Desired Knowledge	Passing arguments allows us to send useful information
Exploration of Target Domain (Programming) Procedure	
Precondition	A method with input parameters.
Required Action	Input parameters are used to solve problem.
Postcondition	Problem is solved appropriately.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Taking an Order.
Precondition	An order has details regarding it.
Required Action	Use those details to prepare the order properly.
Postcondition	The order details allowed the order to be what the customer expects.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	Instructions that take in information.
Required Action	Use the information that was sent to the instructions.
Postcondition	The information allows the instructions to produce a useful result.
Constraints	None.

Table A.20: Only Changing Values as Parameters - Logging in

Identification of Analogy Context	
Misconception	Necessary known values are declared as parameters
Desired Knowledge	Only values that may change execution are needed
Exploration of Target Domain (Programming) Procedure	
Precondition	A method with parameters.
Required Action	Arguments provided are used to complete method.
Postcondition	Method completes and returns successfully.
Constraints	Parameters are only values a user must provide.
Exploration of Source Domain Procedure	
Domain	Logging in.
Precondition	Visiting a service you need to log into.
Required Action	Providing only a username and password to access account.
Postcondition	Input information is validated for successful login.
Constraints	Additional information about you does not need to be provided even if the service knows it.
Analysis of Common Structural Elements	
Precondition	A set of instructions requiring input information.
Required Action	Use of those inputs to complete the instructions.
Postcondition	Successful completion of instructions.
Constraints	Inputs are only information that is not known and needed.

Table A.21: Checking versus Assigning Equality - Saving in Snap!

Identification of Analogy Context	
Misconception	One equal sign used to determine equality condition
Desired Knowledge	Conditions of equality and assigning equality are different
Exploration of Target Domain (Programming) Procedure	
Precondition	A conditional assessing equality.
Required Action	Type == to compare two values.
Postcondition	Values are appropriately assessed for equality.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Saving in Snap!
Precondition	Trying to save to computer not cloud.
Required Action	Use the “export” command to save to your computer.
Postcondition	The xml is appropriately saved to your computer.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A situation where one symbol has two meanings.
Required Action	A second symbol is used for one meaning in a specific context.
Postcondition	The situation resolves correctly.
Constraints	None.

Table A.22: Processing Time of Switch Statements - Ordering from a Menu

Identification of Analogy Context	
Misconception	A Switch Statement “Saves Time”
Desired Knowledge	The computer still takes the same amount of time to evaluate
Exploration of Target Domain (Programming) Procedure	
Precondition	A switch statement.
Required Action	Case that matches to the switched value.
Postcondition	Actions conducted based on case.
Constraints	Each case is considered in order.
Exploration of Source Domain Procedure	
Domain	Ordering from a Menu.
Precondition	A restaurant with a menu of numbered items.
Required Action	An item that you know you want to order but do not know its number.
Postcondition	Ordering and eating the item once you find it.
Constraints	You must review each item to see if it is the item you want to order. Once you find the item you stop.
Analysis of Common Structural Elements	
Precondition	Comparison of several elements to one element.
Required Action	Identical element located.
Postcondition	Actions conducted based on match.
Constraints	Comparison requires each element to be reviewed in order.

Table A.23: Ranged Comparison of Doubles - Parking

Identification of Analogy Context	
Misconception	Doubles compared using exact equality
Desired Knowledge	Doubles may have round-off errors and need ranged comparison
Exploration of Target Domain (Programming) Procedure	
Precondition	A double value.
Required Action	Using a fuzzy or epsilon comparison.
Postcondition	The double is evaluated despite imprecision.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Parking.
Precondition	A stall we want our car to fit safely into.
Required Action	Sometimes our positioning is a little too far to one side or the other.
Postcondition	Our car is still parked in the stall despite not being perfectly centered between the lines.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A value with some imprecision.
Required Action	Checking if the value is approximately right.
Postcondition	The value is compared despite being inexact.
Constraints	None.

Table A.24: Using Methods to Compare Contents - Books

Identification of Analogy Context	
Misconception	Strings compared with ==
Desired Knowledge	Comparison of contents should be done with .equals or .compareTo
Exploration of Target Domain (Programming) Procedure	
Precondition	Two Strings or other reference types.
Required Action	Use .equals to compare not ==.
Postcondition	Information inside is compared not locations.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Books.
Precondition	Two copies of a book one at home and one at the library.
Required Action	Compare the books based on properties to determine similarity.
Postcondition	The books are compared for what book they are not for where they are.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	Two pieces of information stored at locations.
Required Action	Comparison to check the information not the location.
Postcondition	Information is compared based on expected value.
Constraints	None.

Table A.25: First True Condition - A Road Trip

Identification of Analogy Context	
Misconception	An if-elseif structure will identify the best-fitting case
Desired Knowledge	The structure identifies the first case that resolves to true
Exploration of Target Domain (Programming) Procedure	
Precondition	An if-else if-else chain.
Required Action	First condition that fits is followed.
Postcondition	All other conditions in chain are ignored.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	A Road Trip.
Precondition	Several pull offs along the road.
Required Action	The first that has a gas station is chosen.
Postcondition	All other pull offs are ignored before and after a choice is first made.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	Several related possibilities.
Required Action	The first that fits is the one chosen.
Postcondition	All other possibilities are ignored.
Constraints	None.

Table A.26: Comparison Pointing toward Choice - Grocery Shopping

Identification of Analogy Context	
Misconception	compareTo returns which String comes first
Desired Knowledge	compareTo returns a number based on which String comes first
Exploration of Target Domain (Programming) Procedure	
Precondition	Two strings.
Required Action	calling compareTo.
Postcondition	An integer that “leans” left, middle, or right.
Constraints	Strings are ordered based on integer’s lean of negative 0 or positive.
Exploration of Source Domain Procedure	
Domain	Grocery Shopping.
Precondition	Two apples.
Required Action	Setting both apples on each end of a scale.
Postcondition	The scale leans left middle or right.
Constraints	The apple that is lighter is the one the scale leans upward toward.
Analysis of Common Structural Elements	
Precondition	Two pieces of information.
Required Action	A comparison for which comes first.
Postcondition	A value leaning in the direction the first piece is in.
Constraints	Information is ordered during comparison.

Table A.27: Own Version of Instance Variables - People

Identification of Analogy Context	
Misconception	Objects all have the same value for instance variables
Desired Knowledge	Each object has its own version of the instance variables
Exploration of Target Domain (Programming) Procedure	
Precondition	Several objects.
Required Action	Each object has its own set of instance variables.
Postcondition	Changes to one objects instance variables do not affect other objects of the same type.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	People.
Precondition	A group of people.
Required Action	Each person has their own properties like age, height, personality, grade, etc.
Postcondition	One person's properties changing does not cause everyone else's values to change.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	Several things created from a single template.
Required Action	Each thing has its own set of properties based on the template.
Postcondition	One thing's property changes do not affect other things of the same type.
Constraints	None.

Table A.28: Nonstatic Implicit - Board Games

Identification of Analogy Context	
Misconception	The implicit parameter is named somewhere within the method
Desired Knowledge	By making the method nonstatic the implicit parameter is required when called
Exploration of Target Domain (Programming) Procedure	
Precondition	An instance method.
Required Action	An object calls the instance method.
Postcondition	In calling the method the object doing it is known.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Board Games.
Precondition	A piece on the board requires a player to move it.
Required Action	A player must move the piece.
Postcondition	By moving the piece the player whose piece that is is known.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	An action which requires something to do it.
Required Action	A thing must do the action.
Postcondition	By doing the action the thing doing it is known.
Constraints	None.

Table A.29: Encapsulating from User - Ordering Food

Identification of Analogy Context	
Misconception	Encapsulation is limited to variables
Desired Knowledge	Encapsulation can be used on anything a user does not need to interface with
Exploration of Target Domain (Programming) Procedure	
Precondition	A class with many variables and methods.
Required Action	A method is called that uses hidden variables and methods.
Postcondition	The method still completes successfully.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Ordering Food.
Precondition	Ordering an item at a restaurant.
Required Action	The food is ordered but the process of cooking it and special ingredients may be hidden.
Postcondition	Your food is still prepared and given to you successfully.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A process with many actions and pieces of information.
Required Action	The process executes but actions and information it does may be hidden from public.
Postcondition	The process still completes successfully.
Constraints	None.

Table A.30: Constructors Don't Return - Model Kits

Identification of Analogy Context	
Misconception	Constructors have a return type
Desired Knowledge	Constructors have no return type
Exploration of Target Domain (Programming) Procedure	
Precondition	A class.
Required Action	The constructor is called.
Postcondition	The object is initialized when the constructor is called no information is returned.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Model Kits.
Precondition	A model kit is opened.
Required Action	The instructions are followed.
Postcondition	The model is created as part of the process. But the process will not return any other tangible information.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A template for creating something.
Required Action	The process for creating the thing is called.
Postcondition	The thing is created as part of the process the process does not give back a value.
Constraints	None.

Table A.31: Constructors Don't Define Methods - Setting up a Device

Identification of Analogy Context	
Misconception	Constructors have methods within them
Desired Knowledge	Constructors can call methods within them but the class has the methods not the constructor
Exploration of Target Domain (Programming) Procedure	
Precondition	A constructor.
Required Action	The constructor is called.
Postcondition	An object is initialized.
Constraints	The constructor cannot contain method definitions.
Exploration of Source Domain Procedure	
Domain	Setting up a Device.
Precondition	A new computer or phone is being set up.
Required Action	Information and preferences can be set.
Postcondition	The device is configured based on the set up process.
Constraints	The setup process used certain actions but was not were they were defined - we can do privacy control after setup too!.
Analysis of Common Structural Elements	
Precondition	A process for settting up a thing when it is created.
Required Action	The process can set up values and do actions.
Postcondition	A thing is set up based on the process being run.
Constraints	The process cannot define other processes within it only do them.

Table A.32: Instance Variable Setup - Manufacturing

Identification of Analogy Context	
Misconception	Created constructor does not do anything
Desired Knowledge	Constructor allows for instance variables to be set up for object
Exploration of Target Domain (Programming) Procedure	
Precondition	A class template.
Required Action	Use constructor to prepare object's instance variables.
Postcondition	Object is ready for use when initalized.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Manufacturing.
Precondition	A factory line is used to create something.
Required Action	Before it is shipped final details are prepared so it is ready to use.
Postcondition	The item is ready to be used at the end of creation.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A template for creating something.
Required Action	Use the final step of the creation to get the thing ready for use.
Postcondition	Thing that is created is fully prepared to be used.
Constraints	None.

Table A.33: Void Return - Calling a Friend

Identification of Analogy Context	
Misconception	Void methods cannot do actions
Desired Knowledge	Not returning does not mean nothing happens
Exploration of Target Domain (Programming) Procedure	
Precondition	A void method.
Required Action	Modify a reference type defined outside method.
Postcondition	Changes to reference type are seen beyond void method scope.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Calling a Friend.
Precondition	Calling a friend who is long distance to chat.
Required Action	Sharing stories and future plans with your friend.
Postcondition	Friends action perspective and knowledge are altered even though nothing physically changed.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A set of instructions that return no physical results.
Required Action	Actions occur that modify results nonphysically.
Postcondition	Change occurs despite no result being returned.
Constraints	None.

Table A.34: Final Position at Length-1 - A Hallway

Identification of Analogy Context	
Misconception	Index values are zero to the length of the array
Desired Knowledge	Length - 1 is the final index value
Exploration of Target Domain (Programming) Procedure	
Precondition	A defined array in a looping control flow.
Required Action	Start at position zero and increment by one.
Postcondition	The number of values is iterated through when length-1 is observed.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	A Hallway.
Precondition	Leave a hotel room and stand in the hallway with rooms along one side.
Required Action	Your door is zero steps away. Each step takes you to another door.
Postcondition	It will take doors-1 steps to reach the end.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	Being at start point of a collection of elements.
Required Action	Incrementing to each element one movement at a time.
Postcondition	The amount of movements required is number of elements - 1.
Constraints	None.

Table A.35: Array Value is Memory Reference - Campus Directory

Identification of Analogy Context	
Misconception	Array value is its contents
Desired Knowledge	The value of an array is a reference to its location in memory
Exploration of Target Domain (Programming) Procedure	
Precondition	An initialized array.
Required Action	Using the array's name to reference it.
Postcondition	Obtaining the memory address of the array.
Constraints	The information is at the memory address but the array's value is the memory address.
Exploration of Source Domain Procedure	
Domain	Campus Directory.
Precondition	A person you need to collect information from.
Required Action	Looking up that person in the campus directory.
Postcondition	Obtaining an office number for that person.
Constraints	The person has the information but the records gave you the person's location.
Analysis of Common Structural Elements	
Precondition	Multiple pieces of information stored at a location under a name.
Required Action	Using the name associated with the information.
Postcondition	Getting the location the information exists at.
Constraints	The information and location are distinct.

Table A.36: Indexes are Integers - Looking Out a Window

Identification of Analogy Context	
Misconception	Indexes can be values other than integers
Desired Knowledge	Data must be meaningfully read by whole offsets
Exploration of Target Domain (Programming) Procedure	
Precondition	An initalized array.
Required Action	Accessing an element of the array using indexing.
Postcondition	Obtain the element in the array at that position.
Constraints	Position must be a whole number to represent offset from start.
Exploration of Source Domain Procedure	
Domain	Looking Out a Window.
Precondition	Sitting in a room with a number of windows.
Required Action	Something is visible out a specific window.
Postcondition	Walk to that window to view the thing.
Constraints	If we are between windows we may be looking at nothing but a wall.
Analysis of Common Structural Elements	
Precondition	Multiple pieces of information stored at a location under a name.
Required Action	Accessing pieces of information by their position in the set.
Postcondition	Obtaining the piece of information at that position.
Constraints	Position must describe exact starting point of information.

Table A.37: Arrays are Fixed Size - Storing Items

Identification of Analogy Context	
Misconception	An array's size can be changed later
Desired Knowledge	Arrays have a set size determined when they are initialized
Exploration of Target Domain (Programming) Procedure	
Precondition	An array of set size.
Required Action	Values can be contained in the array.
Postcondition	Values can be fit up to the size of the array.
Constraints	Array size cannot change.
Exploration of Source Domain Procedure	
Domain	Storing Items.
Precondition	A container with specific dimensions.
Required Action	Several items can be contained inside.
Postcondition	The container can be filled up to its specified size.
Constraints	The size of the container cannot change.
Analysis of Common Structural Elements	
Precondition	A container of set size.
Required Action	Storage of elements in that container.
Postcondition	Container can be filled to limit.
Constraints	Container cannot change size.

Table A.38: Arrays Do Not Call Methods - Measuring a bookshelf

Identification of Analogy Context	
Misconception	Array length is a method call
Desired Knowledge	Arrays cannot call methods they are not objects
Exploration of Target Domain (Programming) Procedure	
Precondition	An array.
Required Action	Access the "length" property of array.
Postcondition	Information about property is accessed.
Constraints	The array cannot do methods but has variables.
Exploration of Source Domain Procedure	
Domain	Measuring a bookshelf.
Precondition	A bookshelf.
Required Action	Use a tape measure to access dimension details.
Postcondition	Information about the dimension property is accessed.
Constraints	The bookshelf cannot do actions but has dimensions.
Analysis of Common Structural Elements	
Precondition	A collection of elements that cannot do actions.
Required Action	Request a property related to it.
Postcondition	Property information can be accessed.
Constraints	The collection cannot "do actions", but has properties.

Table A.39: Array Length before Use - Bag Shopping

Identification of Analogy Context	
Misconception	Array does not need to be given a size before use
Desired Knowledge	An array must have a size before it can be used
Exploration of Target Domain (Programming) Procedure	
Precondition	A declared array.
Required Action	Must initialize size before adding elements.
Postcondition	Elements can be added after size is initialized.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Bag Shopping.
Precondition	Needing a bag with specific dmensions.
Required Action	We must know the dimensions to buy a bag otherwise we can't fit our items.
Postcondition	With the dimensions known we can add our items to a bag created with the correct size.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A container that is a fixed size.
Required Action	Must know its size before we can put things into it.
Postcondition	Once size is known container can hold elements.
Constraints	None.

Table A.40: Reference Types Create Aliases - Pets

Identification of Analogy Context	
Misconception	Setting two arrays equal duplicates the contents of one to the other
Desired Knowledge	Assigning an array to point to another reference is aliasing but there is only one array with two names
Exploration of Target Domain (Programming) Procedure	
Precondition	Two distinct arrays.
Required Action	One array is set equal to another.
Postcondition	Both array names are now referring to the same array.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Pets.
Precondition	Two dogs have collars with their distinct homes.
Required Action	One dog is adopted into the other's home.
Postcondition	Both dogs collars now reference the same home.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	Two value names referencing distinct locations.
Required Action	One name is assigned to another name's location.
Postcondition	Both names are now referring to the same location.
Constraints	None.

Table A.41: Same Elements are Not Always the Same Array - Sports Teams

Identification of Analogy Context	
Misconception	Arrays containing the same elements are the same array
Desired Knowledge	Having the same contents does not mean that two things exact at the same location - they are similar but distinct
Exploration of Target Domain (Programming) Procedure	
Precondition	Two distinct arrays with the same elements.
Required Action	Determining if the array locations are the same.
Postcondition	The elements are equal but the locations are not.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Sports Teams.
Precondition	Identical sets of gear in different player's homes.
Required Action	Determine a set of gear not based on contents but on location.
Postcondition	The gear is the same but the home it is in is not.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	Two collections of elements with the same elements in different locations.
Required Action	Determining if there is only one collection and one location.
Postcondition	Even if the elements are the same different locations are distinct collections.
Constraints	None.

Table A.42: Extensible Code Design - Getting Dressed

Identification of Analogy Context	
Misconception	Hardcoded end values based on test cases will always work
Desired Knowledge	Code should be designed extensibly for many scenarios
Exploration of Target Domain (Programming) Procedure	
Precondition	Code is being written.
Required Action	Code is designed for any reasonable test case.
Postcondition	The code will still work in all cases.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Getting Dressed.
Precondition	You have an order for putting on clothes.
Required Action	This order works despite any unexpected weather.
Postcondition	You will still be clothed no matter the weather!.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A plan is being created.
Required Action	Plan is designed relative to all applicable scenarios.
Postcondition	The plan will work in any scenario.
Constraints	None.

Table A.43: Brackets and Control Flow - Working

Identification of Analogy Context	
Misconception	Loops do not need brackets
Desired Knowledge	Brackets ensure appropriate control flow for which instructions belong to the loop
Exploration of Target Domain (Programming) Procedure	
Precondition	A loop.
Required Action	Beginning and end brackets around loop statements.
Postcondition	Loop knows appropriate steps to repeat.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Working.
Precondition	An average work day.
Required Action	Indication of when working hours begin and end.
Postcondition	You show up for work on time and complete your tasks appropriately.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	Several steps to repeat.
Required Action	Indication of where repetition begins and ends.
Postcondition	Correct steps can be repeated appropriately.
Constraints	None.

Table A.44: Check Length Before Comparison - Buying clothes

Identification of Analogy Context	
Misconception	In comparison of String characters chosen String to loop through does not matter
Desired Knowledge	Length of String should be checked before characters
Exploration of Target Domain (Programming) Procedure	
Precondition	Two strings to compare.
Required Action	Check if both strings are same length.
Postcondition	If length is the same compare characters.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Buying clothes.
Precondition	Two similar shirts that are on sale.
Required Action	Check if both are the right size first.
Postcondition	If they are check which appears to be in better condition by comparing characteristics.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	Two collections of elements to compare.
Required Action	Check if there is the same number of elements first.
Postcondition	If length is same compare elements.
Constraints	None.

Table A.45: For Loop Condition - Crossing the Street

Identification of Analogy Context	
Misconception	Loop condition is “go until”
Desired Knowledge	Loop condition is “go while”
Exploration of Target Domain (Programming) Procedure	
Precondition	A set of instructions contained within a loop construct.
Required Action	Creating a condition that continues repetition WHILE it is TRUE.
Postcondition	A loop that repeats the steps inside UNTIL the condition is FALSE.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Crossing the Street.
Precondition	You start at one end of the street.
Required Action	While you have not reached the other end you stay alert.
Postcondition	Once you are NOT crossing the street you stop being constantly alert.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A scenario where steps will be repeated.
Required Action	Condtions to continue doing the steps under.
Postcondition	Steps being repeated appropriately while conditions hold true.
Constraints	None.

Table A.46: ArrayList Uses Size - Club enrollment

Identification of Analogy Context	
Misconception	Arraylist has length method
Desired Knowledge	Arraylist method is size
Exploration of Target Domain (Programming) Procedure	
Precondition	An arraylist.
Required Action	Use the size method.
Postcondition	Number of elements is returned.
Constraints	Arraylist can change number of elements.
Exploration of Source Domain Procedure	
Domain	Club enrollment.
Precondition	A club with several members.
Required Action	Request the size of membership.
Postcondition	Number of members is returned.
Constraints	Club members may join and leave.
Analysis of Common Structural Elements	
Precondition	A collection of elements that can change size.
Required Action	Request the size of the collection.
Postcondition	The current number of elements is returned.
Constraints	Number of elements in collection is not fixed.

Table A.47: Immutability of Strings - Engraving

Identification of Analogy Context	
Misconception	The contents of a String can be changed
Desired Knowledge	Strings are immutable. We make new Strings when we set Strings equal to something else
Exploration of Target Domain (Programming) Procedure	
Precondition	A String.
Required Action	The String's name is assigned to a new string as needed.
Postcondition	The original String did not change.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Engraving.
Precondition	A plaque.
Required Action	If the inscription must be changed we need to make a new plaque.
Postcondition	The original plaque did not change.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A piece of information that does not change with a name.
Required Action	We give the same name to new information as needed.
Postcondition	The original information did not change.
Constraints	None.

Table A.48: Execution after Catch - Fire

Identification of Analogy Context	
Misconception	Code ceases execution after a catch block
Desired Knowledge	Code will continue executing any actions after the catch block
Exploration of Target Domain (Programming) Procedure	
Precondition	An exception occurs.
Required Action	The exception is handled.
Postcondition	Normal execution can resume at the point after the handler.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Fire.
Precondition	A fire is occurring.
Required Action	Firefighters handle the fire.
Postcondition	Normal behavior can continue after the fire is put out - but not in the place the fire occurred.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	An emergency situation.
Required Action	The situation is handled.
Postcondition	Normal actions can resume after the situation is handled but not the actions that caused the emergency.
Constraints	None.

Table A.49: Catch Hierarchy - Safety Response

Identification of Analogy Context	
Misconception	Exception handlers must match exactly
Desired Knowledge	Exceptions can be caught if they are part of the catch's hierarchy
Exploration of Target Domain (Programming) Procedure	
Precondition	A specific exception occurs.
Required Action	A catch block for a superclass exists.
Postcondition	The catch can still handle the exception.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Safety Response.
Precondition	Someone is choking.
Required Action	Any person knowing the heimlich can help even if they are not safety personnel.
Postcondition	The situation is able to be handled by any perosn in a larger subset of people.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A specific emergency situation.
Required Action	A situation handler for a broader scenario.
Postcondition	The situation can still be handled by the broad spectrum handler.
Constraints	None.

Table A.50: Closing Writeable Files - Parades

Identification of Analogy Context	
Misconception	Printwriter should always finish writing to file
Desired Knowledge	Without close method buffer may not complete write to file
Exploration of Target Domain (Programming) Procedure	
Precondition	A printwriter.
Required Action	The close method is called.
Postcondition	Any information left in buffer is written.
Constraints	If close is not called leftover buffer text may not be written.
Exploration of Source Domain Procedure	
Domain	Parades.
Precondition	Parade members handing out candy.
Required Action	Members are told when the parade has concluded.
Postcondition	Any leftover candy is distributed.
Constraints	If not told when parade is ending candy might be leftover at the end that wasn't handed out.
Analysis of Common Structural Elements	
Precondition	A means of sending bundles of information out.
Required Action	Means is told when information sending has completed.
Postcondition	Any leftover information is sent out.
Constraints	If not told when ending leftover information can be lost.

Table A.51: Try Container - Pandemic

Identification of Analogy Context	
Misconception	Only first instance of possible exception needs to be in a try block
Desired Knowledge	Any code that could be affected by the exception must be in a try block
Exploration of Target Domain (Programming) Procedure	
Precondition	Code that can throw an exception.
Required Action	All code that the exception can affect is in a try block.
Postcondition	The code should appropriately handle exceptions.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Pandemic.
Precondition	A contagious disease is noted.
Required Action	Any thing that could spread the disease is quarentined and sanitized for safety.
Postcondition	The contagious disease should be handled and not spread.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A potentially dangerous situation.
Required Action	Anything that could be affected has cautionary measures enacted.
Postcondition	The situation should be appropriately handled.
Constraints	None.

Table A.52: Catch and Handle - Safety Protocols

Identification of Analogy Context	
Misconception	Exceptions that should be handled are thrown
Desired Knowledge	Handling exceptions requires catching them
Exploration of Target Domain (Programming) Procedure	
Precondition	A method with an exception.
Required Action	Catch the exception.
Postcondition	Execution can continue.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Safety Protocols.
Precondition	A lockdown.
Required Action	Safety personnel will deal with the situation and give an all clear.
Postcondition	After the all clear business as usual can continue.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	An emergency situation.
Required Action	Dealing with the emergency.
Postcondition	Normal actions can continue.
Constraints	None.

Table A.53: Throw and Catch - Sickness

Identification of Analogy Context	
Misconception	Same exception is both thrown and caught
Desired Knowledge	Throwing and catching the same thing is not useful
Exploration of Target Domain (Programming) Procedure	
Precondition	A method with an exception.
Required Action	Choose to throw OR catch.
Postcondition	If catch handle if throw other method handles.
Constraints	Handling and not handling does not make sense.
Exploration of Source Domain Procedure	
Domain	Sickness.
Precondition	A family member is unwell.
Required Action	Determining whether to care for them at home or take them to the doctors.
Postcondition	If staying at home you handle the situation at the doctor's they handle it.
Constraints	You and the doctor are not helping the family member simultaneously.
Analysis of Common Structural Elements	
Precondition	An emergency situation.
Required Action	Deciding to handle or hand off the situation.
Postcondition	Choosing impacts the next course of action.
Constraints	A decision must be made to continue logically.

Table A.54: Immediate Loop Break - A Gathering

Identification of Analogy Context	
Misconception	break leaves an entire nested loop structure
Desired Knowledge	break only leaves the most immediate loop containing it
Exploration of Target Domain (Programming) Procedure	
Precondition	A loop hits a break.
Required Action	The immediate loop is exited.
Postcondition	Any outer loops are not exited.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	A Gathering.
Precondition	You decide to leave a room when a conversation is unpleasant.
Required Action	You exit the room.
Postcondition	You are still inside the building or property.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A scenario that must be exited from.
Required Action	The immediate situation is left.
Postcondition	Any larger-scale scenarios are still existed within.
Constraints	None.

Table A.55: Ragged 2D Arrays - Calendars

Identification of Analogy Context	
Misconception	2D arrays are always uniform height and width
Desired Knowledge	2D arrays can be ragged
Exploration of Target Domain (Programming) Procedure	
Precondition	A 2D array.
Required Action	Each contained array can define its own length.
Postcondition	The 2D array is ragged.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Calendars.
Precondition	A 12-month calendar.
Required Action	Each month can contain a different number of days.
Postcondition	Not all months are the same exact size.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A collection housing multiple collections.
Required Action	Each collection can have a distinct length.
Postcondition	Not all collections are uniform in size.
Constraints	None.

Appendix B

Revised CS1 OPAL Analogies

Table B.1: Revised A.3: Scanner Storage - White Elephant

Identification of Analogy Context	
Misconception	Scanners stores input to variables automatically
Desired Knowledge	Obtained values must be stored immediately or they are lost
Exploration of Target Domain (Programming) Procedure	
Precondition	An initialized Scanner.
Required Action	Call next methods to obtain input.
Postcondition	Scanner has current input at time of next call.
Constraints	Information is only held at time of next call, and discarded after.
Exploration of Source Domain Procedure	
Domain	White Elephant Exchange
Precondition	A pile of presents.
Required Action	Do some action to get a new present.
Postcondition	You have the one present you obtain after the action.
Constraints	If another action happens, you will always get a new present unless you have acted in a way that allows you to ”hold” your present.
Analysis of Common Structural Elements	
Precondition	A means of obtaining information.
Required Action	Obtain information one piece at a time.
Postcondition	Current piece of information is shown when obtained.
Constraints	Information shown is lost unless stored.

Table B.2: Revised A.4: Multiple Scanners - Office Assistants

Identification of Analogy Context	
Misconception	Multiple scanner creation for single input source
Desired Knowledge	Only one scanner per location is necessary
Exploration of Target Domain (Programming) Procedure	
Precondition	An initialized Scanner.
Required Action	Invoke next methods on the same scanner to obtain input.
Postcondition	Scanner will obtain further input.
Constraints	Additional Scanners initialized to the same location may cause resource access issues.
Exploration of Source Domain Procedure	
Domain	Office Assistants
Precondition	Assistant in an office.
Required Action	Assistant is asked to complete a task using the copy room's resources.
Postcondition	The assistant does the requested job.
Constraints	If multiple assistants are asked to do similar tasks in the same room, they may complete them, but they may also run out of resources, become confused, or repeat/miss steps trying to divvy up work.
Analysis of Common Structural Elements	
Precondition	A means of obtaining input from a location.
Required Action	Continuing to utilize the same input means.
Postcondition	Obtain further input from that location.
Constraints	Multiple obtainers at same location can cause confusion.

Table B.3: Revised A.6: Primitive Wrappers - Making Sandwiches

Identification of Analogy Context	
Misconception	Primitives and their wrapper classes are identical
Desired Knowledge	Primitives and reference types are distinct
Exploration of Target Domain (Programming) Procedure	
Precondition	A situation where a primitive that has a wrapper class is needed.
Required Action	If only value is needed, primitive is used.
Postcondition	Value is accessed simply.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Making Food
Precondition	A sandwich and a lunch box.
Required Action	If you are going to eat the sandwich now, you don't need to store it in the lunchbox.
Postcondition	Easy access to the sandwich without additional steps or overhead.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	Element with value, and an element that is representative of the value element.
Required Action	When only value is needed, element with value is used .
Postcondition	Simplest access to value is carried out.
Constraints	None.

Table B.4: Revised A.33: Void Actions - Safety Deposit Box

Identification of Analogy Context	
Misconception	Void methods cannot do actions
Desired Knowledge	Not returning does not mean nothing happens
Exploration of Target Domain (Programming) Procedure	
Precondition	A void method passed a reference argument.
Required Action	Modify aspects of passed reference type's information.
Postcondition	Changes are seen beyond method's scope.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Safety Deposit Box
Precondition	You and your friend share a safety deposit box, and you have placed an item for them in it.
Required Action	Your friend accesses the box and locates the item.
Postcondition	Changes your friend makes to the item before returning it to the box for you will be there when you later check the box.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	Being given access to a location with items.
Required Action	Change aspects of the items at location.
Postcondition	Changes are visible to anyone accessing that location.
Constraints	None.

Table B.5: Revised A.45: For Loop Condition - Setting the Table

Identification of Analogy Context	
Misconception	For loop condition is “go until”
Desired Knowledge	For loop condition is “go while”
Exploration of Target Domain (Programming) Procedure	
Precondition	A for loop.
Required Action	A starting value and condition that is true while between the first and last value.
Postcondition	The loop executes the correct number of times.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Setting the Table.
Precondition	A circular table for a meal with many guests that must be set.
Required Action	Choosing a spot to begin setting the table, and setting each place while there are still utensils and spots remaining.
Postcondition	The correct number of places are set at the table.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A scenario where steps will be repeated a certain number of times.
Required Action	Where to begin, and a check for repeating steps that allows for repetition while it stays true.
Postcondition	The steps are repeated the correct number of times.
Constraints	None.

Table B.6: Revised A.51: Try Container - Nuclear Radiation

Identification of Analogy Context	
Misconception	Only first instance of possible exception needs to be in a try block
Desired Knowledge	Any code that could be affected by the exception must be in a try block
Exploration of Target Domain (Programming) Procedure	
Precondition	Code that can throw an exception.
Required Action	All code that the exception can affect is in a try block.
Postcondition	The code should appropriately handle exceptions.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Nuclear Radiation.
Precondition	An unusual substance that emits toxic nuclear radiation is located.
Required Action	Any objects it touched that could also have radiation are contained.
Postcondition	The radiation will not spread and the situation is handled.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A potentially dangerous situation.
Required Action	Anything that could be affected has cautionary measures enacted.
Postcondition	The situation should be appropriately handled.
Constraints	None.

Table B.7: Revised A.52: Catch and Handle - Safety Protocols

Identification of Analogy Context	
Misconception	Exceptions that should be handled are thrown
Desired Knowledge	Handling exceptions requires catching them
Exploration of Target Domain (Programming) Procedure	
Precondition	A method with an exception.
Required Action	Catch the exception.
Postcondition	Execution can continue.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Safety Protocols.
Precondition	A bank robbery.
Required Action	Safety personnel lock down the bank to assess and handle the situation.
Postcondition	”After the all clear business as usual can continue”.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	An emergency situation.
Required Action	Dealing with the emergency.
Postcondition	Normal actions can continue.
Constraints	None.

Table B.8: Revised A.53: Throw and Catch - Home Repairs

Identification of Analogy Context	
Misconception	Same exception is both thrown and caught
Desired Knowledge	Throwing and catching the same thing is not useful
Exploration of Target Domain (Programming) Procedure	
Precondition	A method with an exception.
Required Action	Choose to throw OR catch.
Postcondition	If catch handle if throw other method handles.
Constraints	Handling and not handling does not make sense.
Exploration of Source Domain Procedure	
Domain	Home Repairs.
Precondition	The basement is flooding.
Required Action	Determine whether to handle the situation yourself or call a professional.
Postcondition	If you choose to handle it you do not call the professional to deal with the situation.
Constraints	You and the professional are not handling the situation simultaneously.
Analysis of Common Structural Elements	
Precondition	An emergency situation.
Required Action	Deciding to handle or hand off the situation.
Postcondition	Choosing impacts the next course of action.
Constraints	A decision must be made to continue logically.

Appendix C

Instructional Team Analogies

Table C.1: Revised A.28: Nonstatic Implicit - Nametags

Identification of Analogy Context	
Misconception	The implicit parameter is named somewhere within the method
Desired Knowledge	By making the method nonstatic the implicit parameter is required when called
Exploration of Target Domain (Programming) Procedure	
Precondition	An instance method.
Required Action	An object calls the instance method.
Postcondition	In calling the method the object doing it is known.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Nametags.
Precondition	An employee is needed to do some action in a store.
Required Action	The employee does the action and you are assisted.
Postcondition	By looking at their nametag you know who it was that helped you.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	An action which requires something to do it.
Required Action	A thing must do the action.
Postcondition	By doing the action the thing doing it is known.
Constraints	None.

Table C.2: Objects and Non-primitives - Classroom

Identification of Analogy Context	
Misconception	Something that's not a primitive cannot have a “has-a” relationship to another object
Desired Knowledge	An object can have non-primitive components
Exploration of Target Domain (Programming) Procedure	
Precondition	Multiple classes in a system.
Required Action	Creating an object as an instance variable for another object.
Postcondition	Our object contains other objects within it.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Classroom.
Precondition	A room and students.
Required Action	Filling the room allows it to contain many students.
Postcondition	Our room will hold the students within it.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A complex element containing many smaller elements.
Required Action	The smaller elements are described as part of the complex element.
Constraints	None.

Table C.3: Objects and Non-primitives - Cars

Identification of Analogy Context	
Misconception	Something that's not a primitive cannot have a “has-a” relationship to another object
Desired Knowledge	An object can have non-primitive components
Exploration of Target Domain (Programming) Procedure	
Precondition	Multiple classes in a system.
Required Action	Creating an object as an instance variable for another object.
Postcondition	Our object contains other objects within it.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Car.
Precondition	The car body and wheels.
Required Action	Building the car requires putting many wheels to the body.
Postcondition	Our car will have four wheels as part of it.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A complex element containing many smaller elements.
Required Action	The smaller elements are described as part of the complex element.
Postcondition	The complex element and smaller elements relationship is defined.
Constraints	None.

Table C.4: Alias vs Parameter Locations - Discord

Identification of Analogy Context	
Misconception	Modifying a parameter's aliasing affects the original argument sent
Desired Knowledge	The alias is not the same as the parameter
Exploration of Target Domain (Programming) Procedure	
Precondition	Having a reference type variable in one scope and passing it to another scope.
Required Action	Changing the value of the variable within the other scope.
Postcondition	Changes to the value aren't seen outside the scope.
Constraints	If you don't connect any aliases created in the other scope to the first scope then those can't be seen outside the one scope.
Exploration of Source Domain Procedure	
Domain	Discord.
Precondition	Having a Discord account and being a member of a server.
Required Action	Changing your nickname within a server.
Postcondition	Name change is only seen in the server not the account name.
Constraints	If you don't track all of your server names under your account name people in other servers won't know the differnt names you have.
Analysis of Common Structural Elements	
Precondition	The location of an item is replicated from one context to another.
Required Action	The location in the replicated context is changed to a different location.
Postcondition	The location the original context looked at is not changed.
Constraints	If information is not relayed between contexts location redirection is not tracked.

Appendix D

Data Slip Noted Analogies

Table D.1: Constructor Parameters - People

Identification of Analogy Context	
Misconception	N/A
Desired Knowledge	Arrays are a collection of elements where we can access a single specific element via index
Exploration of Target Domain (Programming) Procedure	
Precondition	An array with many indexes.
Required Action	Obtain an array element by indicating the index to access.
Postcondition	The element at that index in the array is accessed.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Trains.
Precondition	A train with many boxcars.
Required Action	Obtain specific cargo by indicating the boxcar to access.
Postcondition	The cargo in that boxcar of the train is accessed.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A collection of several connected elements.
Required Action	Obtain a specific element by indicating its location.
Postcondition	The information at that location is accessed.
Constraints	None.
Details Left on Data Slip	
Topic Week and Indicated Concepts	9: Array Value, Unrelated
Notes	Covered what arrays are in mini-lecture and analogy of train with boxcars being the array and each boxcar being an index

Table D.2: Arrays and Indexes - Trains

Identification of Analogy Context	
Misconception	Objects should all start with the same default characteristics
Desired Knowledge	Constructors can let us set up new characteristics with parameters
Exploration of Target Domain (Programming) Procedure	
Precondition	An object is about to be created.
Required Action	Its characteristics are likely unique, not a ”standard default”, and should use a constructor to be set up.
Postcondition	The object’s details are appropriately defined at instantiation.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	People.
Precondition	A person is about to be born.
Required Action	Their name, birthday, weight, height, and parents are likely unique, not a ”standard default”, and should use the hospital’s data to be set up.
Postcondition	The person’s details were appropriately defined at birth.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	Something is about to be created.
Required Action	It likely has unique characteristics beyond a ”standard default”, and should use that information during set up.
Postcondition	The item or entity’s details are appropriately defined when it is created.
Constraints	None.
Details Left on Data Slip	
Topic Week and Indicated Concepts	8; Weekly Misc (calling constructor with parameters)
Notes	used “people” analogy to explain setting new characteristics with parameters instead of default values

Table D.3: Objects and Classes - Cake Baking

Identification of Analogy Context	
Misconception	An object and a class are the same thing
Desired Knowledge	The created object and the class that defined its creation are different
Exploration of Target Domain (Programming) Procedure	
Precondition	A programmmer wants to make an object.
Required Action	They use the class to inform how the object should be made, and use the constructor to make the object.
Postcondition	The class and the object are not the same thing.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	People.
Precondition	A person wants to bake a cake.
Required Action	They use the recipe to inform how the cake should be made, and follow the steps to make the cake.
Postcondition	The recipe and the cake are not the same thing.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	An item needs to be created.
Required Action	A template is used to inform how the item should be created, and steps are followed to create the item.
Postcondition	The template and the item are not the same thing.
Constraints	None.
Details Left on Data Slip	
Topic Week and Indicated Concepts	8; Return Construct, Construct Setup
Notes	cake baking analogy

Table D.4: Instance Methods - Noun Does Verb

Identification of Analogy Context	
Misconception	Instance method calls do not require an entity to do them
Desired Knowledge	A specific object calls a method with the dot operator allowing the object to access the method
Exploration of Target Domain (Programming) Procedure	
Precondition	We want a method called that requires an instance of a class to do it.
Required Action	We set up the object (noun) to do the method (verb).
Postcondition	The method is called.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Noun Does Verb.
Precondition	We want a task done that requires an item or person to do it.
Required Action	We ask the person or set up the item (noun) to do the task (verb).
Postcondition	The task is executed.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	We want an action done that requires some entity to do it.
Required Action	We request that the entity (noun) do the action (verb).
Postcondition	The action is carried out.
Constraints	None.
Details Left on Data Slip	
Topic Week and Indicated Concepts	8; Weekly Misc
Notes	noun does verb obj . method

Table D.5: Constructor Parameters - Academic Advisers

Identification of Analogy Context	
Misconception	All objects should use the same hardcoded details in their instructor
Desired Knowledge	Creating an object should make use of the parameters provided
Exploration of Target Domain (Programming) Procedure	
Precondition	A class with a constructor.
Required Action	Add parameters to constructors, and use those parameters in the constructor to assign values.
Postcondition	The object will be created with unique values based on the parameters provided.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Academic Advisers.
Precondition	An academic adviser with student reporting software.
Required Action	When they generate a report, they should use the student number you provide to create the report.
Postcondition	The report is unique to you based on the adviser's input.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	Something able to introduce customization into the creation of an item.
Required Action	Details on the customization should be introduced during the creation process.
Postcondition	The item will be unique based on the customization details.
Constraints	None.
Details Left on Data Slip	
Topic Week and Indicated Concepts	8; Topic Not Selected
Notes	use parameter in constructor, not constant string. Analogy: if you go to see the academic adviser and give her your M#, you want her to use your M# and not just the same M# for every student.

Table D.6: Objects and Instance Variables - Parrots

Identification of Analogy Context	
Misconception	Instance variables are known to all objects of class and do not need to be set
Desired Knowledge	Instance variables are set up when an object is constructed, and each object has its own version that only it knows
Exploration of Target Domain (Programming) Procedure	
Precondition	An object.
Required Action	Each object is created with its own set of instance variables, the details of which are not known by other objects unless they interact.
Postcondition	Any specific object has their own instance variables, but all objects don't know each others.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Parrots.
Precondition	A parrot.
Required Action	Each parrot was born with their own name and details, and these are not known by other parrots unless they interact.
Postcondition	Any specific parrot has their own information, but all parrots don't know each others.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	An entity.
Required Action	Each entity is created with its own details, which are not known by other entities unless they interact.
Postcondition	Any specific entity has their own details, but all entities don't know each other's.
Constraints	None.
Details Left on Data Slip	
Topic Week and Indicated Concepts	8; Weekly Misc
Notes	2 Constructor creates values and parrot knows its own info. 1 Instance variables are first declared in class, then set in constructor.

Table D.7: Classes and Objects - Blueprints

Identification of Analogy Context	
Misconception	Classes are objects, or execute their code entirely
Desired Knowledge	A class is a template for objects to be created with
Exploration of Target Domain (Programming) Procedure	
Precondition	An object to be created.
Required Action	To create the object, one must use the class and follow the constructor design.
Postcondition	The object and class are not the same - the class informed how to create the object.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Blueprints.
Precondition	A house to be built.
Required Action	To build the house, one must use the schematics and follow their instructions .
Postcondition	The house and the blueprint are not the same - the blueprint informed how to build the house.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A thing to create.
Required Action	To create the thing, we must use a template and follow its instructions.
Postcondition	The thing and the template are not the same - the template informed how to create the thing.
Constraints	None.
Details Left on Data Slip	
Topic Week and Indicated Concepts	8; Instance Vars, Construct Setup
Notes	class is like blue print name tag analogy

Table D.8: Method Parameters - Relaying Info

Identification of Analogy Context	
Misconception	Parameters do not make a difference to how a method operates
Desired Knowledge	Parameter values should be used to obtain valuable results
Exploration of Target Domain (Programming) Procedure	
Precondition	A method to be called.
Required Action	Parameters provide information that is important to the method's completion.
Postcondition	The method uses that information to complete the task successfully.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Relaying Info.
Precondition	A job must be completed by someone.
Required Action	You provide information that is important to the job's completion.
Postcondition	The person uses that information to complete the job successfully.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	An action to do.
Required Action	Additional input is necessary for the action to achieve desired results.
Postcondition	This input allows the action to be conducted appropriately.
Constraints	None.
Details Left on Data Slip	
Topic Week and Indicated Concepts	8; Weekly Misc (setters)
Notes	parameters get values passed in like when you give info to someone so they can do their job

Table D.9: Equality and Assignment - Boxes

Identification of Analogy Context	
Misconception	Equality checks and assignment are the same thing
Desired Knowledge	Determining if things are equal and assigning them as equal are different actions
Exploration of Target Domain (Programming) Procedure	
Precondition	Variables containing information within them.
Required Action	To check if the variable contents are the same, we look at each and compare them. To make the two variables have the same information, we use one variable to inform what content to assign to another.
Postcondition	Checking if the variables are equal and setting them equal is not the same thing.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Boxes.
Precondition	Boxes containing items within them.
Required Action	To check if the items in the boxes are the same, we look at each and compare them. To make two boxes have the same items, we use one box's items to inform what should go in the other.
Postcondition	Checking if the boxes are equal and putting equal items in the boxes is not the same thing.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	Distinct containers that have elements in them.
Required Action	To check if the containers have the same elements, we look at each and compare them. To make two containers have the same elements, we use one's elements to inform what should go in the other.
Postcondition	Checking if the containers are equal and setting them equal is not the same thing.
Constraints	None.
Details Left on Data Slip	
Topic Week and Indicated Concepts	7; Equals Condition
Notes	putting things in boxes versus taking things out of boxes & comparing them

Table D.10: Returning and Printing - Communication

Identification of Analogy Context	
Misconception	Returning and printing are the same thing
Desired Knowledge	Returning allows a value to be used outside the method, while printing simply displays it
Exploration of Target Domain (Programming) Procedure	
Precondition	A value to send back from a method.
Required Action	You can print information about the value, but only in returning it can another method have and use it.
Postcondition	The calling method has the value after it is returned.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Communication.
Precondition	An item to give to someone.
Required Action	You can talk about the item, but only physically handing it to them allows them to have and use it.
Postcondition	The person has the item after you have handed it to them.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	An entity which must change domains.
Required Action	Actions describing the entity do not allow it to change domain, only physically being sent to the correct domain works.
Postcondition	The new domain now has the entity after it has been sent.
Constraints	None.
Details Left on Data Slip	
Topic Week and Indicated Concepts	6; Return (x2), None (x1)
Notes	print = say return = give back *(Additional: has a note that says explain (connected to analogy)) (referenced in two more notes, total of 3)*

Table D.11: Returning and Printing - Ordering Pizza

Identification of Analogy Context	
Misconception	Returning and printing are the same thing
Desired Knowledge	Returning allows a value to be used outside the method, while printing simply displays it
Exploration of Target Domain (Programming) Procedure	
Precondition	A value to send back from a method.
Required Action	You can print information about the value, but only in returning it can another method have and use it.
Postcondition	The calling method has the value after it is returned.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Ordering Pizza.
Precondition	A pizza to give to a customer.
Required Action	You can describe that the pizza is ready, but only giving it to the customer allows them to have and use it.
Postcondition	The customer has the pizza after you deliver it to them.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	An entity which must change domains.
Required Action	Actions describing the entity do not allow it to change domain, only physically being sent to the correct domain works.
Postcondition	The new domain now has the entity after it has been sent.
Constraints	None.
Details Left on Data Slip	
Topic Week and Indicated Concepts	6; Return
Notes	pizza driver just announces order instead of giving you the pizza

Table D.12: Local Variable Scope - Fish

Identification of Analogy Context	
Misconception	Local variables can be used anywhere
Desired Knowledge	Local variables must be used within their scope
Exploration of Target Domain (Programming) Procedure	
Precondition	A local variable.
Required Action	The variable exists within its local scope.
Postcondition	Outside of that scope, the variable cannot be used.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Fish.
Precondition	A fish.
Required Action	The fish lives in the water.
Postcondition	Out of the water, the fish cannot survive.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	An entity.
Required Action	The entity must exist within its specific habitat.
Postcondition	Outside of that habitat, the entity cannot act.
Constraints	None.
Details Left on Data Slip	
Topic Week and Indicated Concepts	6; Scope
Notes	Local variable is like the fish in the water. Out of water, fish cannot live Out of method. local variable is meaningless

Table D.13: Obtain Scanner Input - Cashiers

Identification of Analogy Context	
Misconception	Information can just be obtained when a means is present
Desired Knowledge	We must do the appropriate actions to obtain information
Exploration of Target Domain (Programming) Procedure	
Precondition	A programmer wishes to obtain a value.
Required Action	The programmer must request the value from the Scanner using methods to obtain it correctly.
Postcondition	The Scanner completes the method and the value is retrieved.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Cashiers.
Precondition	A person wishes to check out with their groceries.
Required Action	A person must find a cashier and check out in order to obtain their groceries legally.
Postcondition	The cashier completes the checkout and the person has their groceries.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	An entity that wants to obtain some elements.
Required Action	The entity must ask for an action to be completed by another entity in order to obtain the elements properly.
Postcondition	The entity completes the action and the elements are obtained.
Constraints	None.
Details Left on Data Slip	
Topic Week and Indicated Concepts	5; Multiscanner
Notes	Needed to call .nextInt on scanner used textbook example & analogy to needing a cashier to checkout

Table D.14: Additional Notes Regarding Analogy Use by Instructional Team Members

Details Left on Data Slip	
Topic Week and Indicated Concepts	13; Except Contain
Notes	used the pre defined analogy and one using COVID-19 and quarantine
OPAL Table	This analogy already exists in Table B.6

Appendix E

Non-CS1 Instructor Analogies

Table E.1: Thread Waiting - Checkout (Customer POV)

Identification of Analogy Context	
Misconception	A thread can always wait on a CV without conditional constraint
Desired Knowledge	As CV signals are lost if not threads are waiting, conditional contraints must be used
Exploration of Target Domain (Programming) Procedure	
Precondition	The thread wishes to wait for permission to take an action.
Required Action	The thread checks if permission has already been granted; if not the thread indicates that it wants permission and waits; if so the thread consumes permission and continues.
Postcondition	The thread only acts when permission has been granted.
Constraints	Mutual exclusive access to the monitor (structural limitation).
Exploration of Source Domain Procedure	
Domain	Checkout.
Precondition	Want to purchase items and needs a clerk available.
Required Action	Check to see if clerk is currently available; if not the person queues; if so the person starts checking out and occupies the clerk.
Postcondition	The person is only able to check out when the clerk is available.
Constraints	??? This is a really difficult constraint that I'm passing up nailing down right now.
Analysis of Common Structural Elements	
Precondition	The procedure requires a resource.
Required Action	The procedure must wait while the resource is not available.
Postcondition	The procedure completes its task with ownership of the resource.
Constraints	??? This is a really difficult constraint that I'm passing up nailing down right now.
Provided Notes	
Notes	This is a major problem that students have to tackle, and I do think I solve it with analogy. However, expressing that "larger" analogous process succinctly is challenging. Am I perhaps trying to phrase too many aspects of the problem at once? Would this work better if I decomposed this into a set of relationships

Additional Notes by Instructor 1

Table E.2: Thread Waiting - Checkout (Clerk POV)

Identification of Analogy Context	
Misconception	A thread can always wait on a CV without conditional constraint
Desired Knowledge	As CV signals are lost if not threads are waiting, conditional contraints must be used
Exploration of Target Domain (Programming) Procedure	
Precondition	The thread wishes to grant permission for another thread to take action.
Required Action	The thread checks if threads are already waiting for permission; if not the thread indicates that permission has been granted; if so the thread signals a thread waiting for permission.
Postcondition	The thread grants permission either indirectly (through state) or directly (through signalling).
Constraints	Mutual exclusive access to the monitor (structural limitation).
Exploration of Source Domain Procedure	
Domain	Checkout.
Precondition	Need to assist customers in checking out purchases.
Required Action	Check to see if customers are currently waiting; if not the clerk indicates that they are available; if so the clerk starts assisting the customer.
Postcondition	The clerk is only able to check out when a customer is waiting.
Constraints	??? This is a really difficult constraint that I'm passing up nailing down right now.
Analysis of Common Structural Elements	
Precondition	The procedure provides a resource.
Required Action	The procedure offers up the resource if no procedure is waiting for it.
Postcondition	The procedure facilitates another's task by giving ownership of a resource.
Constraints	??? This is a really difficult constraint that I'm passing up nailing down right now.
Provided Notes	
Notes	To continue on, I decomposed my initial attempt into two separate perspectives. Both are addressing the same problem and the same misconception but they frame the problem from two different actors/threads. The core misconception relies on the intersection between 4 different branches of control flow between those two different actors/threads.

Table E.3: Shared Memory Pointers - Library

Identification of Analogy Context	
Misconception	Pointers to malloc'd memory may be (productively) stored in shared memory
Desired Knowledge	Pointers to malloc'd memory are meaningless across multiple processes
Exploration of Target Domain (Programming) Procedure	
Precondition	Local memory region (malloc'd) with important data.
Required Action	Copy contents of local memory into shared memory.
Postcondition	Contents can be accessed by any process connected to shared memory.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Library.
Precondition	A set of books at home you wish to donate.
Required Action	Physically transport copies of the books to the library.
Postcondition	Books can be interacted with by anyone in the shared space.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	Information exists local to some private point of reference.
Required Action	Information is copied verbatim from the source to a location with some public point of reference.
Postcondition	Information is available to anyone privy to the public point of reference.
Constraints	None.
Provided Notes	
Notes	I feel like my setup for this is backwards. This is an analogy that illustrates the proper action to be taken but does not seem to target the heart of the misunderstanding: the folley of using local pointers within shared memory.

Table E.4: Shared Memory Pointers - Library (Contrapositive)

Identification of Analogy Context	
Misconception	Pointers to malloc'd memory may be (productively) stored in shared memory
Desired Knowledge	Pointers to malloc'd memory are meaningless across multiple processes
Exploration of Target Domain (Programming) Procedure	
Precondition	Local memory region (malloc'd) with important data.
Required Action	Copy pointer to local memory into shared memory.
Postcondition	Other processes will attempt to access their own local memory at the pointer.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Library.
Precondition	A set of books at home you wish to donate.
Required Action	Leave a note with the library saying ”the books are on my dining room table”.
Postcondition	Books cannot be interacted with by anyone in the shared space because the books are not on other patron's dining room tables.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	Information exists local to some private point of reference.
Required Action	Private point of reference is communicated through some shared resource.
Postcondition	Information is unavailable to others because they do not share the private point of reference.
Constraints	None.
Provided Notes	
Notes	Here is my approach to the contrapositive...

Table E.5: Semaphores and Mutexes - Etiquette

Identification of Analogy Context	
Misconception	Semaphores and mutexes form a ”block” of code
Desired Knowledge	Semaphores and mutexes are commands and can be used freely (intelligently)
Exploration of Target Domain (Programming) Procedure	
Precondition	One or more threads wish to share a criticial section concurrently, without outside threads concurrently sharing.
Required Action	The first thread to attempt critical section entry obtains the mutex lock; the last thread to leave the critical section releases the mutex lock.
Postcondition	The critical section can not be entered by an outside thread until all members of the group leave.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Etiquette.
Precondition	One or more people wish to claim a table.
Required Action	The first member of the group to arrive claims the table; the last member of the group to leave releases the claim on the table.
Postcondition	The table cannot be claimed by another group while it is claimed by this group.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	(Blank).
Required Action	(Blank).
Postcondition	(Blank).
Constraints	(Blank).
Provided Notes	
Notes	This one somewhat broke me. The reason why, I believe, is that I misinterpreted the notion of a ”required action”. I finished it off, but I really don’t know if this one is ”correct”. Also, this falls into a similar trap to the above example. It shows a contradiction to the misconception, but not necessarily an abstracted notion?

Table E.6: Countdown Semaphore - Dining Philosophers

Identification of Analogy Context	
Misconception	N/A
Desired Knowledge	(In the dining philosophers problem) a countdown semaphore represents a limitation of chairs
Exploration of Target Domain (Programming) Procedure	
Precondition	A set of dining philosopher threads, with the mutual exclusion, hold and wait, no pre-emption, and circular waiting properties.
Required Action	Acquiring resources is protected using a countdown semaphore, limiting the number of threads actively participating in the critical section.
Postcondition	A set of dining philosopher threads, with the mutual exclusion, hold and wait, and no pre-emption properties (eliminating circular waiting).
Constraints	The initial value of the countdown semaphore is smaller than the number of threads.
Exploration of Source Domain Procedure	
Domain	Dining Philosophers.
Precondition	A set of dining philosophers, with the mutual exclusion, hold and wait, no pre-emption, and circular waiting properties.
Required Action	The philosophers now require a chair in order to eat, and there are a limited number of chairs.
Postcondition	A set of dining philosophers, with the mutual exclusion, hold and wait, and no pre-emption properties (eliminating circular waiting).
Constraints	The number of chairs is smaller than the number of philosophers.
Analysis of Common Structural Elements	
Precondition	(Blank).
Required Action	(Blank).
Postcondition	(Blank).
Constraints	(Blank).
Provided Notes	
Notes	Anthropomorphism. I decided to go with something accessible and directly taught in the course. In this example, I am specifically describing a programming structure (countdown semaphore) to a collection of chairs. Although now that I'm working through it I'm not sure what the appropriate way of entry would be.

- I1: Overall, I don't see any flaws with the template itself. Even better, I think the structure helped me confront some of my analogy usage and really force myself to formalize what I'm analogizing. And I guess it helped you run across some issues that don't necessarily come about in lower level classes. I do think that there are some points (as you can see in the first constraint group) where the constraint in the target domain may not have a ”reasonable” related constraint. Most of what immediately came to mind to relate the monitor's mutual exclusion constraint were very contrived ideas which seemed outlandish. Would contrived things pull the students away from the analogy, because it makes the source domain seem foreign and awkward? I don't know, but I feel like it's a possibility
- *(Indicated prior to E.6)* I1: Misconceptions v No Conception. As we talked about, sometimes when you miss the mark students don't get it wrong... they just don't get it at all!

Table E.7: Method Scope - House Keys

Identification of Analogy Context	
Misconception	Methods should always be public
Desired Knowledge	Only methods designed to be part of a public interface should be marked as public. Other methods should have a more restrictive access modifier.
Exploration of Target Domain (Programming) Procedure	
Precondition	Methods should always have the largest scope.
Required Action	Methods that require information about implementation and helper methods should be kept private.
Postcondition	Methods that aren't part of the public interface should have the most restrictive access modifier needed to have the program work.
Constraints	Some methods require different degrees of access modifiers.
Exploration of Source Domain Procedure	
Domain	House Keys.
Precondition	A house where everyone has keys.
Required Action	Someone with a key to the house can access every part of it. Some people are just guests, and shouldn't have access to the whole house or things may go missing.
Postcondition	Only give people enough access to do something. Don't give other people access to everything.
Constraints	Some people need more access than others (E.G. a delivery person needs less access than a close friend, who needs less access than residents of the home).
Analysis of Common Structural Elements	
Precondition	An assumption that a student's code will not be used by other programmers.
Required Action	Encapsulation.
Postcondition	A defined public interface and other methods hidden and encapsulated.
Constraints	None.
Provided Notes	
Notes	None

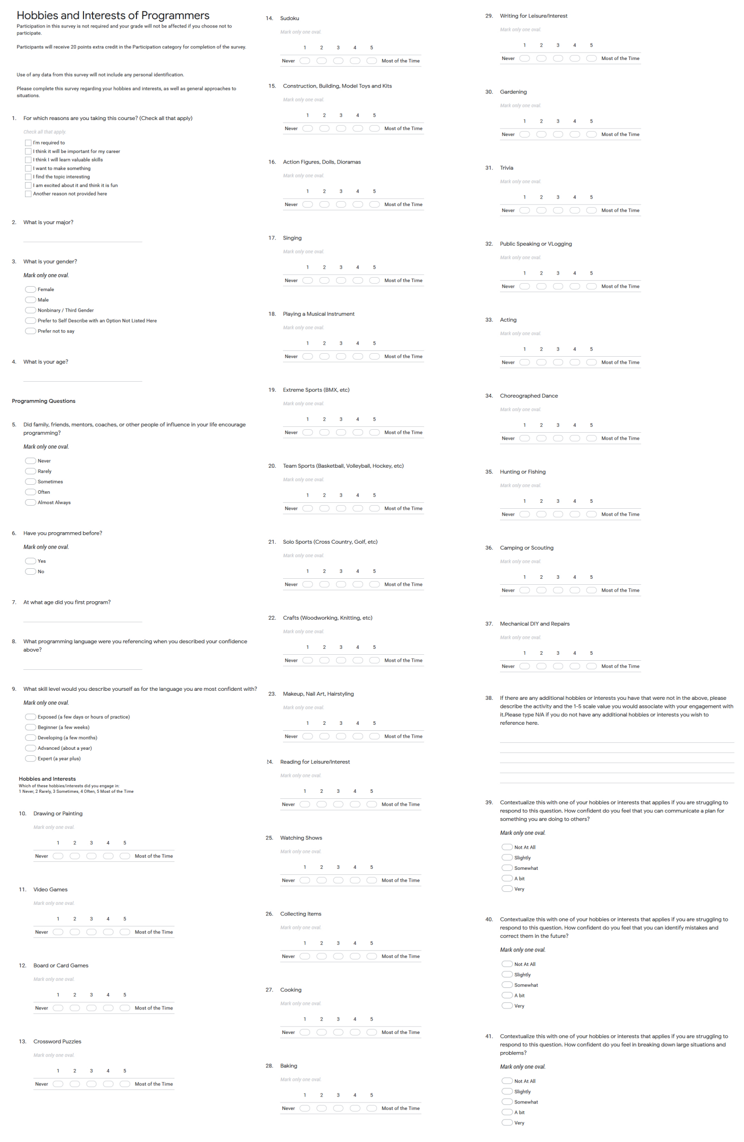
Table E.8: Subclass Implementation - Cars

Identification of Analogy Context	
Misconception	Methods should always be public
Desired Knowledge	Only methods designed to be part of a public interface should be marked as public. Other methods should have a more restrictive access modifier.
Exploration of Target Domain (Programming) Procedure	
Precondition	Subclasses need to re-implement all methods.
Required Action	Only methods that are abstract need to be overridden.
Postcondition	Methods with implementations in the superclass can be left undefined.
Constraints	None.
Exploration of Source Domain Procedure	
Domain	Cars.
Precondition	Cars of similar models, but with different features.
Required Action	Making a brand new blueprint for two cars of the same model, but different features doesn't make sense.
Postcondition	Have a single blueprint, and only implement different variants instead of re-making the blueprint for each configuration of accessory.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	A belief that all methods need to be explicitly defined at the lowest level.
Required Action	Realize that some implementation can be done at a higher level.
Postcondition	While methods can override superclass methods, this is optional and not required.
Constraints	None.
Provided Notes	
Notes	None

Table E.9: Generics - One Size Fits All Clothing

Identification of Analogy Context	
Misconception	It is acceptable to use Object methods and classes that work on objects instead of Generic classes.
Desired Knowledge	Generics should be treated as their own data type and manipulated as such as opposed to casting to a specific type.
Exploration of Target Domain (Programming) Procedure	
Precondition	Methods should treat generics as any other data type most of the time.
Required Action	Treat generics as though they were any other data type, as long as you aren't allocating space or using a constructor.
Postcondition	Casting should only be done with Arrays designed to store Generics
Constraints	If a method compares two objects as opposed to two generics of type E, we can't enforce they are the same type using an object method.
Exploration of Source Domain Procedure	
Domain	One Size Fits All Clothing.
Precondition	A hat with "one size fits all" can fit one person at a time, not every person at a time.
Required Action	A hat enforces only one person uses it at a time.
Postcondition	A hat only belongs to one person at a time.
Constraints	None.
Analysis of Common Structural Elements	
Precondition	Students assume "can handle anything" is the same as a "can handle everything"
Required Action	The distinction between "anything" and "everything" needs to be made, as "anything" can allow for some subtle, important restrictions.
Postcondition	Objects relate to "can handle everything", while generics relate to "can handle anything".
Constraints	None.
Provided Notes	
Notes	This analogy doesn't translate very well into text and the more simple domain issue, but during conversation this helped the student understand the underlying issue since we also touched on the misconception mentioned in the abstracted schema of "everything =/= anything"

Hobbies & Interests Survey



Hobbies and Interests of Programmers

Participation in this survey is not required and your grade will not be affected if you choose not to participate.

Participants will receive 20 points extra credit in the Participation category for completion of the survey.

Use of any data from this survey will not include any personal identification.

Please complete this survey regarding your hobbies and interests, as well as general approaches to situations.

1. For which reasons are you taking this course? (Check all that apply)

   Check all that apply.

   - ☐ I'm required to
   - ☐ I think it will be important for my career
   - ☐ I think I will learn valuable skills
   - ☐ I want to make something
   - ☐ I find the topic interesting
   - ☐ I am excited about it and think it is fun
   - ☐ Another reason not provided here

2. What is your major?

3. What is your gender?

   Mark only one oval.

   - ◯ Female
   - ◯ Male
   - ◯ Nonbinary / Third Gender
   - ◯ Prefer to Self Describe with an Option Not Listed Here
   - ◯ Prefer not to say

4. What is your age?

Programming Questions

5. Did family, friends, mentors, coaches, or other people of influence in your life encourage programming?

   Mark only one oval.

   - ◯ Never
   - ◯ Rarely
   - ◯ Sometimes
   - ◯ Often
   - ◯ Almost Always

6. Have you programmed before?

   Mark only one oval.

   - ◯ Yes
   - ◯ No

7. At what age did you first program?

8. What programming language were you referencing when you described your confidence above?

9. What skill level would you describe yourself as for the language you are most confident with?

   Mark only one oval.

   - ◯ Exposed (a few days or hours of practice)
   - ◯ Beginner (a few weeks)
   - ◯ Developing (a few months)
   - ◯ Advanced (about a year)
   - ◯ Expert (a year plus)

Hobbies and Interests

Which of these hobbies/interests did you engage in:
1 Never, 2 Rarely, 3 Sometimes, 4 Often, 5 Most of the Time

10. Drawing or Painting
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

11. Video Games
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

12. Board or Card Games
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

13. Crossword Puzzles
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

14. Sudoku
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

15. Construction, Building, Model Toys and Kits
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

16. Action Figures, Dolls, Dioramas
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

17. Singing
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

18. Playing a Musical Instrument
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

19. Extreme Sports (BMX, etc)
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

20. Team Sports (Basketball, Volleyball, Hockey, etc)
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

21. Solo Sports (Cross Country, Golf, etc)
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

22. Crafts (Woodworking, Knitting, etc)
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

23. Makeup, Nail Art, Hairstyling
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

24. Reading for Leisure/Interest
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

25. Watching Shows
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

26. Collecting Items
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

27. Cooking
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

28. Baking
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

29. Writing for Leisure/Interest
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

30. Gardening
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

31. Trivia
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

32. Public Speaking or VLogging
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

33. Acting
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

34. Choreographed Dance
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

35. Hunting or Fishing
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

36. Camping or Scouting
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

37. Mechanical DIY and Repairs
    Mark only one oval.
    1 2 3 4 5
    Never ◯ ◯ ◯ ◯ ◯ Most of the Time

38. If there are any additional hobbies or interests you have that were not in the above, please describe the activity and the 1-5 scale value you would associate with your engagement with it.Please type N/A if you do not have any additional hobbies or interests you wish to reference here.

39. Contextualize this with one of your hobbies or interests that applies if you are struggling to respond to this question. How confident do you feel that you can communicate a plan for something you are doing to others?

    Mark only one oval.

    - ◯ Not At All
    - ◯ Slightly
    - ◯ Somewhat
    - ◯ A bit
    - ◯ Very

40. Contextualize this with one of your hobbies or interests that applies if you are struggling to respond to this question. How confident do you feel that you can identify mistakes and correct them in the future?

    Mark only one oval.

    - ◯ Not At All
    - ◯ Slightly
    - ◯ Somewhat
    - ◯ A bit
    - ◯ Very

41. Contextualize this with one of your hobbies or interests that applies if you are struggling to respond to this question. How confident do you feel in breaking down large situations and problems?

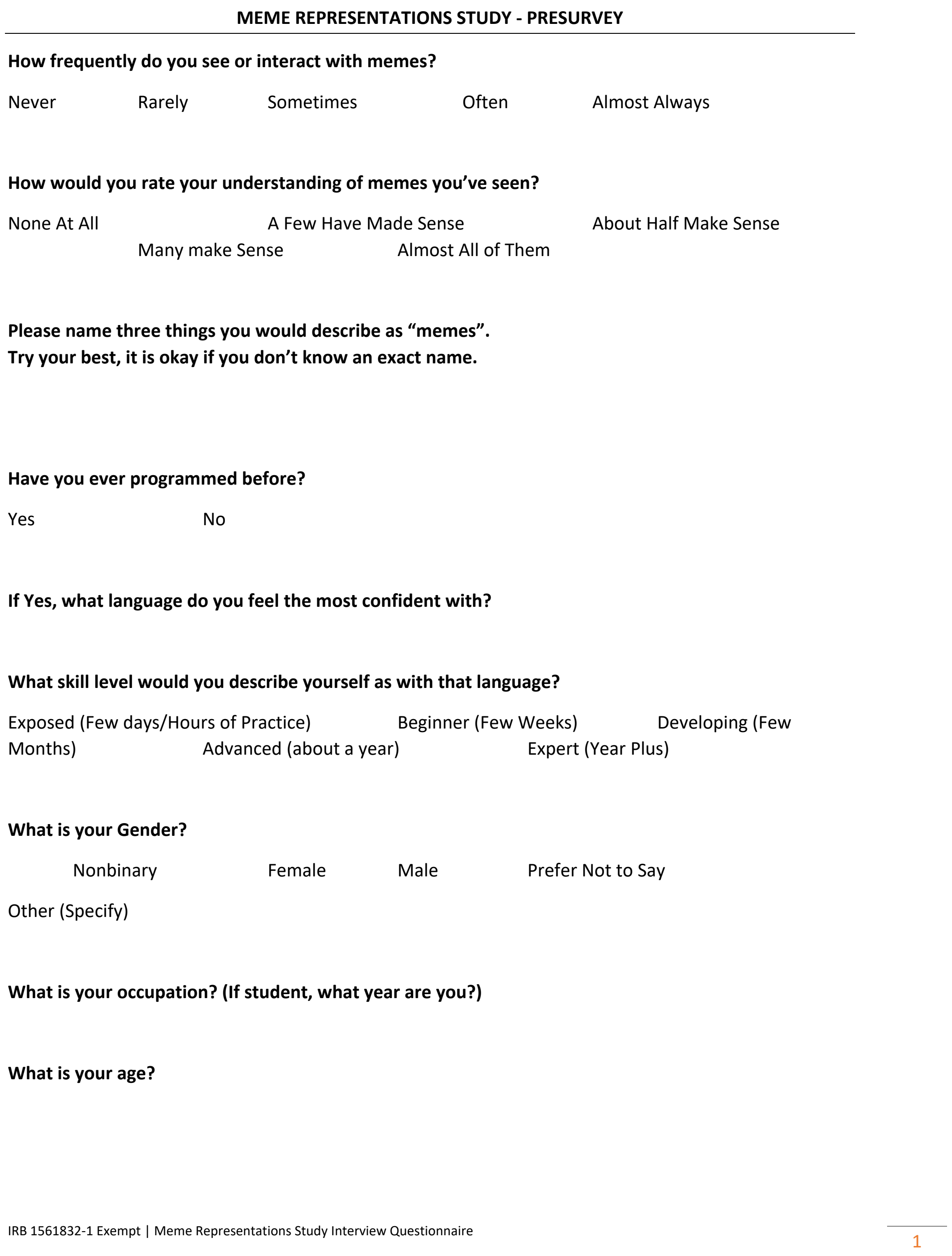
    Mark only one oval.

    - ◯ Not At All
    - ◯ Slightly
    - ◯ Somewhat
    - ◯ A bit
    - ◯ Very

Appendix G

Meme Interview Guide

APPENDIX G. MEME INTERVIEW GUIDE



MEME REPRESENTATIONS STUDY - PRESURVEY

How frequently do you see or interact with memes?

Never Rarely Sometimes Often Almost Always

How would you rate your understanding of memes you've seen?

None At All A Few Have Made Sense About Half Make Sense
Many make Sense Almost All of Them

Please name three things you would describe as "memes".
Try your best, it is okay if you don't know an exact name.

Have you ever programmed before?

Yes No

If Yes, what language do you feel the most confident with?

What skill level would you describe yourself as with that language?

Exposed (Few days/Hours of Practice) Beginner (Few Weeks) Developing (Few
Months) Advanced (about a year) Expert (Year Plus)

What is your Gender?

Nonbinary Female Male Prefer Not to Say

Other (Specify)

What is your occupation? (If student, what year are you?)

What is your age?

IRB 1561832-1 Exempt | Meme Representations Study Interview Questionnaire

1

MEME REPRESENTATIONS STUDY – POSTSURVEY

Do you feel reflecting on the presented memes increased your understanding of what some of these templates are attempting to convey?

Not At All	Slightly	Somewhat	A Fair Bit	A Lot

Do you feel that reflecting on the programming memes impacted your understanding of programming ideas?

Not At All	Slightly	Somewhat	A Fair Bit	A Lot

Would you have revised any of your answers as you went through the process? Did your understanding change for any of the memes or templates as the study went on?

<u>CHANGE BASED ON RESPONDENTS – Ask one for respondent, one reflective</u>

Do you think that understanding aspects of the memes changed your ability to understand the memes that related to programming knowledge?

Did your knowledge of programming impact your understanding of the memes and what they were trying to communicate?

Do you think a programmer who doesn't understand memes might be able to draw meaning?

Do you think someone who understands memes but not programming might be able to draw meaning?

Do you have any questions or closing comments?

APPENDIX G. MEME INTERVIEW GUIDE

MEME REPRESENTATIONS STUDY – GUIDED QUESTIONS

BLANKS QUESTIONS

Have you seen this meme template before?

Do You Have familiarity with any content in this meme?
Characters, text, symbols, gestures?
Describe what you recognize and how you recognize it.

What context do you think this image is creating?
Why might someone use or relate to this image based on what you see?

(If Text) Does the text on the template modify your perception of the image? If so, how?

NONBLANK QUESTIONS

Please describe the situation you think this meme is trying to convey. What does it mean?
Who or what is this meme about?

Who is it intended for? Who could relate to this? What situation might this be used in?

How are you arriving at that understanding based on the images and text shown?

Are any elements such as text or images modifying your understanding when viewing this meme?

Appendix H

Meme Study Quotes

Evidence of "Is This a Pigeon?" Structure

- P7: *bursts into laughter* I know the first time I saw the if statement AI memes INTERVIEWER: So you've seen this one before? PARTICIPANT: Not necessarily this one, but I've seen, other memes comparing AI to if statements, I think the first time I saw it, I didn't really understand AI super well beyond um it being like a buzzword, and so I didn't necessarily get it but then when I used to see it with different memes and different contexts and that connection, I was able to infer both with my newer understanding over time but also just inferring from those different memes and how they were usually used what the connection between those two are so now I like I have a better understanding of what artificial intelligence is and so I you know might joke oh yeah it's like conditionals and different kinds of conditionals um but I think that this the the format and because this is a popular meme, it makes it, even if you didn't understand you know, the connection between artificial intelligence and if statements as like, you know, abstract concepts, I think that would help at least, introduce the relationship of if statements have something to do with AI but they aren't necessarily the same or at least they have some key aspects that can be considered similar even though they aren't INTERVIEWER: So in this case, there's some similarity between these two things but they are - if statements would be more of a subset would you say or? PARTICIPANT: I wouldn't necessarily say they're really a set of one another to me, the way I see it is like a venn diagram between AI and if statement, and there is some overlap in that diagram in terms of like, artificial intelligence using if statements but I think it's more of if statements just being a representation of a conditional and AI using conditionals to evaluate things using searches um and playing games and stuff like that, you know basing things like making deci- I guess making decisions. If statement is the introduction of if you've made a decision, this is how we execute, and AI is figuring out how to make decisions, so I think that there's that disconnect and you know there's a disconnect with this meme, because why would it be in this meme format if there isn't a disconnect?
- P10: Because it's not inaccurate but also not accurate but the con-everything in a computer works off the basis of is this true or is it not true everything in that AI which is this like super like high level topic that's like the future kind of thing um can be boiled down to the fact that uh in the computer it's still just um, if true or if not true kind of thing.
- P12: [...] I think I've seen this exact one actually, I I find it funny. Uh if statements can model a like intelligent decision based purely on like a a yes or no, but to the untrained eye it could seem like your program has artificial intelligence, because it does something you know it it it's thinks for itself, even though it's obviously not doing it

- P27: Yeah or they just think it's funny because there's like a small similarity that they share that they try and say that they're the same as a joke.
- P7: Like I've seen a lot of these for like programming jokes but like if I send someone this meme with a programming joke on it and they don't understand programming, well they're gonna see like, oh well he's confused but they're not gonna understand why you know, "oh the semicolons matters" or whatever they're not gonna understand the joke itself, but they're gonna understand the meaning behind his face and say "oh there's a joke, but I just don't have the education to understand it"
- P21: So programmer if statement is this AI? Um so it's it's a programmer um and then they're saying they're looking at an if statement and asking is this an AI or is this AI um, because if the if statement is a butterfly, even if you if you understood meme culture even if you didn't understand programming you would be able to understand that an if statement obviously like would not be generally considered AI but like, programmers would like to act like it because they say "is this AI?"
- P29: Yeah, I think, like, we saw literally I had that moment with the uh, dog comic thing that you showed me, I don't know what any of what they were talking about was, but I understand the joke because I recognize the format, so my friends would see "oh, is this AI?", if statement - they would probably come to their own conclusion that an if statement must not be AI.

Evidence of "It's Free Real Estate" Structure

- P17: I feel like there needs to be like another meme before it that like says what he's talking about INTERVIEWER: Okay so you think there needs to some more content around this for it to make sense PARTICIPANT: Mhm
- P7: I think the exists keyword is what introduces the idea of that's what's gonna be the real estate is, because the RAM isn't actually, doing a particular action, in this template, where it would be the one "taking" the free real estate, sort of like how like, the cat sees the box, and is the one responding, or um, I'm doing something to empty the box, is the difference that makes me, even though I'm the other acting party, not the real estate, the box is, but in this one, the RAM is just existing, there isn't an, specific action, so we know that, you can assume that, JVM is the one taking that RAM.
- P24: See I don't know what JVM is but I see the Java marks and I think it means that um like JVM takes up a lot of your um RAM. Like the cat it's like moving into your RAM.
- P9: So the it's free real estate they corrupt the guy's eyes with Java so um so well I don't necessarily unders- I don't know what JVM is, um, but I I can imagine that that it's that the that the RAM is the RAM and asterisks exists means it's there, that you *pauses* *chuckles* that that like that that the person um or JVM sees that it's an untapped market in that they can you know capitalize on that.
- P20: So this is making fun of how Java likes to uh use as much RAM as it can and so it's just saying that like oh, you have RAM in your computer, and then Java's like "ooh! free real estate let me just use it" INTERVIEWER: Alright, how did you get Java out of this? PARTICIPANT: Uh first, Java Java oh the eyes or the Java logo and then JVM is Java Virtual Machine or something like that.

Evidence of "Ight Imma Head Out" Structure

- P12: When you declare a variable inside whatever curly bracket uh idea and then you close whatever that it is it you can't use it anymore because it would be out of scope to use it anywhere else um so the variable itself can't be used and it's it's headed out like Spongebob
- P15: So yeah it's not actually physically heading out, but it's done being like it can't be referenced later past that bracket or like okay I'm done.
- P21: Local variables when they see a *laughs* I don't, I forget what the names of that is but I know the name I use for them definitely isn't right. um INTERVIEWER: That's okay PARTICIPANT: I, I so this is me er pretty sure that this is referring to that local variables stop when they see that it's closed because it won't be the local instance anymore, I think that's the right word for it. But that they won't be used anymore, so it's saying that aight, I'm just gonna leave cause like they're done being used.
- P17: The curly brackets um local variables can't like leave outside of the curly brackets right? So like then you can't access them after that. I think. I don't, I don't remember.
- P22: Oh the curly bracket doesn't look like it's a good thing so maybe that would cause an error with the local variable? INTERVIEWER: What do you think that error might be? PARTICIPANT: I don't...I don't know why it makes the local variables not...work it looks like? Whatever kind of variable that would be it's not for you

Evidence for "Does Your Dog Bite?" Structure

- P6: I- forgot that arrays started at zero *chuckles* INTERVIEWER: Okay, so, so you're saying that you forgot that arrays start at zero so - PARTICIPANT: Yeah so, they start at index zero so that's like, if you wanna start yeah arrays if you wanna do the very first element in the array it's array element er array index zero, not one WOOPS
- P8: Does your dog bite no array indexing starts at one um, I don't know a whole lot about arrays maybe some people think they start at zero? Or maybe some other specific number, and that's how they do it, but, maybe this is like the actual true textbook way to do it, so they're hurt by that fact and they don't wanna give up their ways.
- P9: I don't know exactly like, exactly what array indexing is however, I have heard that like, for some reason, in coding languages the that the start is one and it doesn't start at zero but then for some reason it would make more sense for things to start at zero, I don't know why it would make more things to make sense at zero but I think but like I know that like some in the in the programming community or CS community have a large gripe with array indexing starting at one, because allegedly it does not make much sense *chuckles*
- P14: so in this one the dog is actually saying something that's wrong so um the because I think indexing it starts at zero not one it like it's almost like cringey to the person in green because they know that's wrong.
- P18: It could also be that he's saying that and that's painful to the other person because it's false so it could kind of be either way I guess as far as like he's either saying that because it's true in some like particular language and the person in the green shirt's like "nooo that's

so counterintuitive" or he's saying something false and the person's like "oh, did I teach you nothing?" *laughs*

- P30: Like the pain or the hurt is that is them starting at like zero er wait er no like okay so this is like, because we know indexing starts at zero so like what hurts is him saying it's like indexing starts at one because like we know it's not true.
- P8: He's the subject of this whole meme here, um and a modification of him modifies who the meme is really tailored for, so like, like I said you could put like, boomer on his shirt, and then have like an Okay Boomer here, and then he'll be all hurt by it.
- P23: To me the dog is MatLab INTERVIEWER: *chuckles* PARTICIPANT: And it can hurt when you get to a language where array indexing starts at zero, where it can be really hard to remember INTERVIEWER: So MatLab does or doesn't start at zero? PARTICIPANT: It starts at one. INTERVIEWER: Okay, so the dog is MatLab because he's telling a, a truth for how he works? PARTICIPANT: Yes. INTERVIEWER: And that's, hurtful to the green guy...because? PARTICIPANT: Because maybe the green guy's not using MatLab, maybe the green guy doesn't

Evidence for "Who Would Win?" Structure

- P21: And the left is really like complicated and like, complex and well thought out, and then if you just like threw one little tiny wrench in with like a single character with no second one to finish it off it would just make everything crash to a halt.
- P29: Curly bracket with unmatching end curly bracket so they take and make into like a funny internet slang term whereas the computer program with millions of lines of code that's very very official, very like you know, I guess official is the same word I was gonna go towards. And so in this meme, it's saying that because this curly bracket doesn't have a end curly bracket, it could completely mess up the entire millions of lines of code in that program.
- P9: That it that the code does not work as it's supposed to because the curly boy does not have an ending curly boy um so part of the part of the what makes it humorous is that you have curly boy and also curly boy is spaced out so it's kind of like, there's there's different like, there's different fonts and a way that words or way that words and phrases are are kind of set up where if that makes it more humorous as part of like the internet culture type thing that definitely helps but the basic premise I think is that like that you have you set up this entire system and then you you don't have an ending curly bracket or curly boy to have it work so you know
- P8: The no friend thing I'm assuming that it's a closed bracket here, like a closed curly boy, um, and I think if you're doing like a big line of code and you accidentally forget to add the closing bracket to something, it won't run. I think? INTERVIEWER: Okay, alright, so who would win in that situation? PARTICIPANT: I guess the curly boy would win. Because of yeah, all that code would mean nothing if this isn't complete I think.
- P15: They forgot to put the last bracket on the end of their code so like there's this huge program there kind of implied like millions of lines of code it's probably not gonna be, it'll maybe hundreds or something, thousands, but they just forget to put the closing brace so it won't compile or yeah they're getting errors and they need to figure out why that one curly

boy with no friend, just curly boy funny name for the bracket, and with no friend, there's no closing bracket.

- P15: Yeah well, *sighs* I guess that comes from experience like programming forgetting something like that, because like it's implying some sort of competition between the two, like who would win, so here it's like, you got this computer program millions of lines of code, and then how is it competing with one curly boy and, I guess yeah the relationship I implied is that that bracket is in the line of code and that's so the bracket is gonna win in this situation because it won't compile and that's kind of the humor.

- P17: It's just as bad as having an error, because it probably won't work. INTERVIEWER: Why do you think millions of lines of code might not work? PARTICIPANT: Because it would take way too long for the computer to process and then like it would probably just get lost, I don't know it might eventually work but it would take forever like way way too long to be useful.

- P23: I don't really see the comparison between the computer program with a million lines of code versus a single curly bracket, they seem like they're in two totally different categories. And what are they trying to win? It just doesn't make a lot of sense to me. INTERVIEWER: Okay so why would these be in kind of different categories? What do you think uh, each, what are some of the comparisons that you've tried to run through? PARTICIPANT: So I'm sort of comparing like computing power, but a curly bracket has no computing power in and of itself, and the other comparison that I'm looking at is uh, the idea of which one is more likely to crash, but that doesn't make sense with "who would win" - "who would lose" seems better if it's looking at which one is more likely to crash. INTERVIEWER: *chuckles* Which one is more likely to crash, if you were going down that route? PARTICIPANT: Probably the single curly bracket that doesn't have another one with it but there's nothing that says that the computer program with the million lines of code doesn't have - there's nothing that says it can compile.

- P29: Um, I don't think it changed interpretation of the structure, I think in the specific context like, if I go out and I go to Twitter after this and I find a um who would win meme, I'm not gonna be like oh kinda like that curly friend meme I saw earlier you know, but when I was looking at that curly friend, I think the knowledge I have of programming definitely helped with that specific meme if that makes any sense?

- P24: Like on the left panel would be a computer and then on the other side would be like a picture of like a clipart caterpillar and it would say like "who would win - a computer or one buggy boy" or something

Evidence for "Gru's Plan" Structure

- P16: *chuckles* So this is like a class and oh just because you fixed one thing doesn't mean it's gonna fix everything else or it doesn't mean you're gonna your program's gonna run correctly so it's like okay listen something's wrong in our program so the programmer's go through they're like okay we found the bug we fixed the bug and then they're like oh but now we have two bugs and then they and then they just kinda and then they like fix those two bugs and then it's like oh no they have three bugs and they're like "oh my God when is it ending" so it *laughs* this particular meme just expresses the frustration of programmers and how if they have a bug and they fix it, it doesn't mean that their program is fixed.

- P22: *laughs um, you thought you found the bug and that you fixed it, but then it's just like, there it keeps creating more bugs and you don't notice it until like after the fact and then you're like shoot.
- P29: Ohhoho I hate when this happens. Okay so, the first two he's talking about there's a bug in the program, so he's like we find the bug and then we fix the bug and now we have two bugs, and he's kind of like wait a minute, well now we have three bugs, because this happens sometimes when you fix, when you think you fixed one bug, but really just starts a chain reaction for two other bugs, and then soon there's three other bugs

Evidence for "Whatcha Got There?" Structure

- P9: So from what I infer from the meme format is that you is that the programming professors do not want you to use global variables so right and the opposite of global variables I'm assuming is local variables so I think that that you that that the programming professors want you to use and er want you to define and then use local variables while while you might *chuckles* might be wanting to use global variables so it's it's kind of the you know it's not necessarily a it's and he says a smoothie because like you know, nothing to see here um it's it's kind of the it's the idea that like you that you're using something that you're not supposed, like you're not supposed to have an ostrich you're not supposed to have a global variables in your code.
- P11: Well these are the global variables so they're probably the ones that don't disappear they're kind of like cheat codes in programming and then the programming professors are probably like you're not supposed to use those or like "whatcha got there" INTERVIEWER: So you're not supposed to use that, I like how you said cheat codes, so you, there's an implication that you shouldn't use what's on the ostrich kind of thing like? PARTICIPANT: Or like, yeah, I guess. INTERVIEWER: Because the professors are asking about it? PARTICIPANT: Or like it's not the most acceptable thing
- P12: Um this one I'm gonna I'm gonna take a guess I I haven't really worked with globals yet, but I would assume that it's not very uh uh um accepted when you're a learning student to just use global variables everywhere because it might make the task easier for you and it might not be good practice when you're working in the industry. Um so programming professors could try to deter you from using global variables, but you may use them anyway to make your job a lot easier, that's what Spencer is trying to hide - Spencer is the gentleman in the bottom panel. That's what he's trying to hide while, you know, toting his smoothie.
- P14: So it seems like global variables are bad? Like, [inaudible] programming professors, but like me or like any student to the programming professor kind of tries to brush past it.
- P16: Okay so I haven't used a global variable yet but I'm assuming it's like a variable that you can make like can be used in any like scope or or any method or something um and that's I guess what the professors are looking at as kind of like cheating so like oh like "whatcha got there" and they're like "oh you know, a smoothie" like trying to trying to deflect on what what the actual problem is.
- P17: Is there a reason you're not supposed to have global variables? INTERVIEWER: I can't answer that right now, but you are saying that is there a reason you're not supposed to have global variables, so you're indicating that you don't think you should have them right?

PARTICIPANT: Yeah, I'm gonna assume that you're not supposed to have global variables so you're just making up some stuff to tell your professor.

- P22: Are global var-are global variables like a no no? It must be a no no because you don't want to tell your professor that you have them is all I'm....
- P25: And there's a global variable and then you're obviously trying to hide that under the rug. INTERVIEWER: Do you have any idea why you might try to hide that under the rug? PARTICIPANT: Uhhhh, I don't think it's good to have I don't quite know why it's not good to have yet. Um but it's probably not a good thing to have.
- P8: Um, so, if a local vari-well, if I'm right about the local variable being one that you defined, maybe global variable is *pauses* I have no idea *laughs* INTERVIEWER: No it's okay PARTICIPANT: Eh, global variables are just bad I guess and you shouldn't use them but a student did, maybe for like a shortcut, or they just didn't know the right way to do it, I don't know.
- P21: Usually with the thing they would just like they would just point out the thing directly. But like, if if my programming professor saw that I was using global variables they would say like you know why don't why are you using global variables instead of "whatcha got there" or like if a cop pulled me over for an illegal racecar, they would be like "this is an illegal car you can't have this" so yeah I I I think it's I think it's more pointing pointing out the fact that like the top group doesn't want you to have the thing at the bottom or whatever or the thing at the bottom is bad for them and then you kind of like pointing away from it.
- P7: As a programming student, even though I rarely use global variables, you know I'm constantly hearing my professors complain about people using global variables, so I understand that there is a um, I guess negative relationship between programming professors and global variables? Or at least, you know, programming professors don't like global variables so that's gonna be the thing that they pay attention to um, I can assume that me is the student, based on them submitting something.
- P19: *laughs* Okay so funny because we're not supposed to use global variables and um he knows that, they know that, but they see him using global variables and they're calling him out on it.
- P20: Ummm no. I don't I guess there would be but I don't think it would be an actual smoothie. INTERVIEWER: What what might they be doing instead? What would be kind of superimposed as the smoothie here? PARTICIPANT: I guess they'd be like showing other parts of the code or like ignoring that fact and like scrolling past it er what's on your computer.

Evidence for "Rollsafe" Structure

- P8: You don't want an infinite loop to happen because then your program never accomplishes it's goal I think um, but, I I think if it gets to the point where the goal is supposed to be reached and that doesn't happen then you get an error message, so if it never reaches the point where it's supposed to check if the goal is reached then you can't get that error message when it obviously doesn't work um but an infinite loop isn't the answer because they're not gonna get anywhere if they do that.

- P9: So that the way that you avoid this potential problem is by causing another problem *chuckles*
- P14: Like technically you can't get an error message like running it in a loop but that's obviously not fixing your problem because there's something with it when it's stuck in a loop.
- P15: Their code is working, but it's in an infinite loop so it's not working and that's kind of the those two things playing against each other is the humor in this one INTERVIEWER: So it's working but not working - how can, how which part of it is working? PARTICIPANT: They're not getting the error message, but there's still a bug.
- P22: *chuckles* Well it's still like, you don't *laughs* you don't want there to be an error even if it is in your infinite loop, so he's like trying to hide it. He thinks he's being slick, even though it's not the best idea.
- P24: Again it's getting rid of one problem by creating another one. INTERVIEWER: So what problem did we get rid of? PARTICIPANT: The error message, but now we have an infinite loop
- P28: The idea is not really bright, but it is true. But there's really no like, real application where you would use this because if you can't really you can't get an error if it's stuck in an infinite loop because the code's just going on and on and on and it's not going to throw an error so it's good that it's not throwing an error but it's also a huge problem that it's an infinite loop but it's again just juxtaposing the idea of like oh like, it's not throwing an error that must be good, but it's also bad because it's in an infinite loop.

Humor, Self-Identification, and Sense of Belonging

- P11: INTERVIEWER: Oh boy, did that one hit close to home? *laughter* PARTICIPANT: *laughter* yes! INTERVIEWER: Why is that? *chuckles* PARTICIPANT: So like the dog is saying array indexing starts at one and usually when it's started at zero, but then if you do that it messes up the whole program so array indexing starts at one is like ahh
- P12: Oof. INTERVIEWER: *chuckles* oof. PARTICIPANT: That that one would wound me actually, that would actually make me [inaudible] um so I don't know any languages that the array index starts at 1 but, there could be? Uh array indexes typically start at zero in the languages I'm familiar with and having them start at one would definitely put a wrench in a lot of people's um brains for a bit to try to like rework that so that's why the gentleman is crying because if what the dog said was true it would throw everyone for a loop for a good while.
- P15: Well it's wrong first of all. INTERVIEWER: *laughs* PARTICIPANT: *chuckles* um, yeah, um so yeah there's a programmers all argue like arrays start index start at one or indexes start at zero, I'm I'm on team zero so this is targeted at me, and I'm supposed to be the one like, grabbing my arm like "oh no that hurt" but
- P20: This one hurts a little bit because it's arrays starting at 1 that's one reason I do not like MatLab *laughs* like I always forget

- P25: *sharp intake of air* Noooooooo *laughs* do you know that this is what MatLab does? *laughs* It's so bad. INTERVIEWER: You had a very visceral reaction to this. PARTICIPANT: *laughs* because it's MatLab and I hate MatLab! And so it's like it can hurt the dog can hurt you in other ways, is array indexing starts at one and it's terrible
- P25: Oh so they're saying that the the code's not in the it's not even gonna compile if you if they don't have the curly boy doesn't have friends or it's not gonna work. This happens a lot. INTERVIEWER: This happens a lot? PARTICIPANT: Well, uh at least to me, because I forget which curly bracket goes to another, so I'll delete like a whole if statement and then like oop everything's red and Java hates you now.
- P29: Ohhoho I hate when this happens. Okay so, the first two he's talking about there's a bug in the program, so he's like we find the bug and then we fix the bug and now we have two bugs, and he's kind of like wait a minute, well now we have three bugs, because this happens sometimes when you fix, when you think you fixed one bug, but really just starts a chain reaction for two other bugs, and then soon there's three other bugs
- P6: Uh programming professors "uh whatcha got there?" me "global variables" a smoothie *chuckles* yeah, professors always drill it into our heads like don't use global variables but uh-again, they're just so convenient INTERVIEWER: They're so convenient so, why would a global variable be the ostrich in this example?
- P10: Um INTERVIEWER: *chuckles* you smilin' PARTICIPANT: Yeah I've gotten yelled at about this multiple times so
- P8: Which I'm very guilty of, like oh I should be writing man, I spent like six hours last night writing my paper but meanwhile like, refresh Twitter like, went on Discord, or like, you know, something like that. INTERVIEWER: So you said you're guilty of it, have you ever written a book? PARTICIPANT: No, not written a book, but you know like, writing a paper, you know something like that. INTERVIEWER: So this is this can be extensible even though you said that you know that the bottom text belongs to a writer, who's working on a book, we can extend our understanding of it? PARTICIPANT: Yup, because everybody's you know, had to do something some kind of responsibility that they've had to work on, and take time to do, but, maybe they've procrastinated a little too much and that's where the relatable sense comes in.
- P10: Yes and basically it is working on my and just replace book with literally anything else and I think most people would abstract or they kind of get the point.
- P15: Right yeah it's, memes don't need to necessarily apply to real life, it's just kind of um yeah you can figure out the situation because cops illegal racecar, cops try to prevent illegal things so they're gonna obviously ask um me in the meme and they're gonna try to not get in trouble, and you can you can figure out the meaning from the context provided INTERVIEWER: Okay, so you can kind of situate yourself in that context even if it doesn't apply to you? PARTICIPANT: Right
- P25: I was I was just crying, I was like seventeen *chuckles* and I didn't know *laughs* this poor cop, he's like fresh out of the academy he's pulling over this poor seventeen year old girl that's crying in a beat up Chevy Tracker *laughs* at six thirty in the morning *chuckles*
- P26: INTERVIEWER: Okay so you, so you kind of get this meme in general, so PARTICIPANT: Yeah, like parents too, like when you've got something like hidden, I bought *chuckles*

I went to the mall once and I bought a skirt black skirt, and I bought this like crop top shit and I came home with both, my shirt was white, and my skirt was black, and I showed my mom the skirt, and I pulled it out and I showed the skirt and I was like look, and she was like okay and I put it back in the bag and I was like kay bye, and mom was like what's the white thing in the bag, and I said, I don't have a white thing in the bag mom I just bought the skirt *laughs* so we spent like the next like 10 minutes chasing each other around the house until she finally figured out what was in the bag, so it's like one of those things where you know, the one thing I want to show my mom, but I didn't want her to see the other one, so it was a, you know

- P24: *laughs* I've seen this one before, it um it basically like it means that like a single missing curly bracket can throw off your code in Java like if you don't have a pair then it's not gonna compile in the program that runs other programs *laughs*
- P20: I know I've sent this to people like when we were I was leaving a Discord chat and I just sent that as like a picture like see ya I'm leavin'
- P12: I made one of these uh earlier this week INTERVIEWER: Oh, you've made one? PARTICIPANT: Yes.
- P16: I actually just recently saw one today about um online classes and losing friends and I [indecipherable] *chuckles*
- P8: That you don't really want a part of the conversation at hand I'd say it's most handy in like a group chat where somebody's like, talking about some kind of cringe or whatever and you're like "alright, I don't, yeah that's not" so I'mma head out
- P20: Memes are tailored to people or some sort of group and each one has to it depends
- P12: No, however I think the mom index is a good index INTERVIEWER: The mom index? PARTICIPANT: So if my mom can understand the meme, then it's a good meme
- P29: Totally, totally did. INTERVIEWER: Okay do you have any thoughts or comments on that, or just? PARTICIPANT: I think it just makes the context more relatable you know like, I love programming memes I think they're so funny when they're done right. Um, and so it kind of like combines those two worlds, because obviously I'm not going to be in class looking at memes, but then at the same time like, when I do go do it on my own free time, I'm not gonna be talking about programming all the time, so it's like combining those two worlds of like ay I recognize that and it's talking about something I know about so it's kind of like that I guess kind of, I don't wanna say feeling of inclusion, because that's not really the word I'm looking for, but it's kind of like the recognition of oh my gosh there are other people who are like me who they like memes and they like programming and they're making content about it.

Subversion and Career of Metaphor

- P6: Usually but *chuckles* INTERVIEWER: What would usually happen? PARTICIPANT: Usually they repeat the third and fourth panels but honestly this, with this format that's funny that's just as fu-*chuckles* INTERVIEWER: So why why is it funny now that they changed it, the fourth one? PARTICIPANT: Cause it's true with programming like, you find a bug, and you fix it, but then all of a sudden a new bug is occurred and uh, something else has occurred too so you fix one problem only to get like five more.

- P9: So this mm, this is kind of *chuckles* this is this is interesting because the the third and fourth panels aren't the same and usually that, when you when you have something like this, usually the third and fourth panels are the same um but it this does work because the the the fourth panel's worse than the third panel, that the severity is worse um if it were to be we fixed the bug we now have three bugs and then we now have two bugs that kind of makes the joke worse because like then it's what, you're solving the bugs

- P12: *chuckles* uh it's it's very common that when you're debugging a program um you'll create more bugs than you fix um so that's the struggle that Gru is going through he's elaborating his plan and then he's discovered that he's accidentally made two bugs *laughs* oops I didn't even read the last panel *laughs* it *laughs* it it's subverts your your understanding of the of the meme because when I've seen it the third and fourth panel are typically the same um but it adds the comedy because it does seem that bugs kind of multiply out of nowhere, you could change a small thing and it would create a bug somewhere.

- P15: Um it it added to it I'd say, because like um yeah there's just the bugs keep growing, when typically it'd just be like now we have two bugs, he'd look again like oh now we have two bugs, instead he looks over and it's like now we have three bugs, you know like oh I wasn't expecting that you see this meme so much you kind of can almost fill in the fourth panel, looking at the third one. So then when you look and it's something different it will add to the humor. INTERVIEWER: So that's an interesting comment. I've heard a few other participants say the word subversion of the expectation. PARTICIPANT: Yeah, that's a, that's a good word for it. INTERVIEWER: So that, that adds to the humor, when when they expect the template, is broken in some way and then you have to kind of assess why did they do that? PARTICIPANT: Yes.

- P25: Oh nooo, oh so there's a bug, and you fix the bug, oh oh it changed! *laughs* Now we have two bugs, and look at it now there's three. *laughs* Um so they're saying that they fixed one problem and then that problem went to like a few more problems that happened to have children because now there's two bugs and then there were three.

- P28: Um, well this one is kind of making fun of like the meme template itself a little bit too um because usually the last two ones are the same, and he's reflecting on like the same thing that he just like put up on the on the poster board saying like oh yeah I did this, but oh look that's what actually happened, but this one actually has two different um bottom pictures and so it's saying they find the bug they fix the bug and then I guess two new bugs appear just because they fixed that one bug so clearly they're not done with like debugging their code and they look away for like one second and look back and there's a third bug so it's just making fun of usually there's only supposed to be two of the same pictures, but in this one because if a coder looks away for like a second, there's going to be a third bug, kind of makes fun of the template and it's making fun of like, if you fix one bug, you're usually not just done to your coding.

- P29: It changed it a little bit, because I think, usually this third and fourth panel are supposed to be stagnant where he's kind of like, realizing what he just said on the third panel, but I think now the context it's showing that like the situation's getting worse, so it kinda changes how the meme is functioning but you still get a general message across.

- P12: Mhm. Yeah, it's a very subtle change to a format you're familiar with which I feel like can bring new life into a meme a little bit I've seen it happen a lot. INTERVIEWER: Okay so um, sometimes changing the format can be helpful in terms of making the meme feel funny? PARTICIPANT: Oh yeah, of course.

- P10: You end up having one more panel to end that's kind of working around that in some absurd way INTERVIEWER: Is it, is it the same images as any of these panels, or is it an entirely new image that's added for the fifth and sixth panels, so do we replicate any of these er do we get a new thing? PARTICIPANT: It's usually a replication of this one here this, usually a second one of this one INTERVIEWER: Okay so the order is this so we have 3rd 4th and then we replicate the 4th for 5th and the 3rd for 6th? PARTICIPANT: Yes. INTERVIEWER: So it would be here, here, and here? So he's like looking twice and then making the face in the third panel in the last panel, is that what you're saying? PARTICIPANT: Yeah.
- P21: *chuckles* when you deep fry an image it's usually you put like a little bit of like a layer filter er a yellow kind of like layer over it and then like it gets really pixellated um it originates from like when you share a meme over and over and over again by like screenshoting it, it loses quality over time and it kind of gets that effect to it but then people like artificially put that effect into it to like show that there's more emotion I guess and just that it er and it's it definitely comes from a certain type of meme but
- P29: Yeah I feel like, because you see the meme used so many different ways in so many different jokes over and over again, you know, sometimes its just the same joke over and over again and people aren't creative, you just kind of like subconsciously learn what each one means, you know? Like, there's no Twitter post that's like hey guys I'm gonna start a new meme, here's how to use it here's what it means, and anyway here's a blank template you know? Someone just makes it because it's funny. Like I made a meme last night about my D&D campaign. You know, people just make them without any context, and they're still funny, you know?
- P29: It takes a while because it's it's hard because you might see a funny text post but it's not really a meme, a meme is something that's used over and over again in the same format for different context and so once you see - when a new meme starts you don't really know it's a meme yet, because you just see a few people using it, but once it gains momentum and you see more people using it then you start to understand the context of it like oh this is a meme, I have more understanding of what it is now versus if I saw like two people use it I wouldn't really get the understanding.
- P28: Understand like a little bit of like the culture behind it or like how like memes can build off of each other and can interact with each other and all that INTERVIEWER: What does memes like building off of each other and interacting with each other look like? PARTICIPANT: Um you can use multiple different templates to make like a single template from like ones that had already been very popular and another one that's already been very popular kind of mesh them together a little bit um and make one that just like brings both of those ideas together more, like one that's even funnier

Appendix I

CS1 Interview Guide

APPENDIX I. CS1 INTERVIEW GUIDE

VERBAL CONSENT:

I am conducting research on novice programmers and how their models and understanding of programming concepts is built. I am interested in your experiences as a current or past student of the CS1121 Intro to Programming course. The purpose of this research is to understand in greater detail how information from the course is assimilated by students, and what associations they create with the material. Your participation will involve this virtual interview which will last approximately between thirty minutes and an hour. This research has no known risks.

Please know that I will do everything I can to protect your privacy. Your identity or personal information will not be disclosed in any publication that may result from this study. Any data collected from this interview such as notes will be kept as secure as possible.

You have every right to cease participation at any point, or to not wish to answer a question. If you state you do not wish to answer a question, I will move on.

Do you have any questions about the procedures I have just described?

May I obtain your verbal consent that you agree to participate in this interview?

Also, may I record the audio of this interview to better help me transcribe? Any audio files will not be released, they will only be utilized for accurate transcription and notes.

FOR EVERYONE:

Age:
Gender Identity:
Major and Year in School:

"With this work, we're trying to better understand how students think about certain concepts from the introductory programming course. Things like what helps, what doesn't, what is hard, and what is easy. There are no "right" or "wrong" answers, we just want to better understand how you think about this material."

Had you ever programmed prior to CS1121?
What ideas did you have about programming prior to the course?

What ideas or things do you associate with programming and coding?

Have your perspectives and associations with programming and coding changed from when you started? In what ways? What do you think caused that change?

What topics from CS1121 do you find most interesting?

Most boring?

Most difficult?

Easiest?

What topics in programming do you look forward to learning?

What topics were different than what you expected?

What assignments in the course were most interesting?

Most boring?

Most difficult?

Easiest?

What aspects of the course helped you to learn?

Were there any specific resources, discussions, problems, or examples, or things from lecture and lab that stand out to you? Why?

What aspects of the course do you feel could have changed to help you learn even more? In what ways do you feel that would aid you?

What do you feel is the most important thing you have learned from CS1121?

How do you feel what you learned/are learning relates to real world situations and problems?

What are you most proud of from your work in CS1121?

APPENDIX I. CS1 INTERVIEW GUIDE

What would you have done differently if you could go back?

What was one of your most challenging moments in CS1121?

What was one of your strongest “aha” moments, where something clicked, in CS1121?

How do you plan to use what you learned in CS1121 in the future?

Has your work in CS1121 accomplished the result you wanted when you signed up for the course? Why? Has your desired result changed over time?

FOR CS1121 STUDENTS ONLY:

Possible Choices (At least one, two would be great!) - based on submissions:

- Top 1: Week 9, Problem 5 (Arrays)
- Second: Week 8, Problem 4 (Objects)
- Third: Week 6, Problem 4 (Methods)
- Fourth: Week 10, Problem 4 (Loops)
- Last: Week 7, Problem 4 (Conditions)

ABOUT SPECIFIC WEEKLY TOPIC:

How did you feel about the topics from this week?

What made sense? What was confusing?

Did any specific resources, discussions, problems, or examples, or things from lecture, lab or the book clarify these ideas for you? Did they confuse these ideas for you?

Did prior knowledge or outside resources/examples influence any thoughts or feelings you had?

Did you feel any strong emotions when working on assignments and the material for this week?

ABOUT SPECIFIC EXERCISE QUESTION:

“Walk me through your process for this specific exercise question - or if you didn't’ complete it, the process you had while working on this question.”

What thoughts went through your mind as you worked on this particular problem?

What information guided your responses?

What references did you make to lab, lecture, the book and other classroom resources?

What references did you make to prior knowledge or outside resources or examples?

What difficulties did you encounter? How did you think about and approach them?

How did you feel confident that you had overcome them? What made you feel less sure?

How did you finalize your answer and decide it was correct? Or, how did you decide on what you ultimately submitted?

How would your response change now? What have you learned since then? Do you feel more confident or less confident? Would you have changed how you responded? In what ways?

What would you go back and tell yourself now, if you were able to, about working on this particular assignment?

FOR EVERYONE:

Do you have any closing comments or questions?

Appendix J

CS1 Interview Quotes

J.1 CS1 Student Perspectives

Participant 1

- P1: You know I messed around with a couple different languages and there was mostly just, you know I just pick at doing this stuff so there's just like, I take working examples, and I would integrate that into any code that I was trying to do, most of the times it didn't work so, but the times I could get it to work it was just, you know, it was just working examples that I never really came up with you know, it's just, I guess, it's just to learn by example, so the hardest thing I had with that was trying to find uh like um I don't wanna say legit but quality like quality information source to draw from to reference it off of, and that's, that's what I used to do it
- P1: I like like, how in depth we go when we talk about certain subjects in class it's it's been really good like understanding primitives, not primitives, and what each commands do, like, things where the, when the memory's being allocated, like we're setting this up, like, when you explain hey this a line of code, you know, and you're like, verb does this, you know, um, those things make sense to me, because then I can take that code and I can understand what I'm trying to do with it, even if I'm messing up something [...]
- P1: I just like, if you think about when you pick up a training manual from the military it's very short and sweet and right to the point, it's bare bones and that's, and I kind of got used to that I realized that this was, the, I I didn't realize how far away from academics that really uh, you know, and trying to adjust back to that because there is such a contrast between it [...]
- P1: [...] and I have these memory issues and things kind of, I get tripped up really easily and I get confused and frustrated really fast so like I'll be tracking an issue and then all of a sudden something interrupts me and now I I don't even remember what's going on and then I start getting more and more upset and confused and frustrated because it's it's just begun. So, when it comes to like getting back on track it can, you know, and actually learning the course material, I have to step away from everything for a while until everything kind of just starts to calm down and go back into order because it's like I I mean I just you know, it, every, everything is just overwhelmed and overloaded and, having to cover material like course material is really, it's really difficult because it doesn't you know I I can beat my head on that brick wall all day and it doesn't matter, but if I step away from a day or two, then I'm like oh man, I, my brain, got to rest and I can start to remember things and I can remember bits and

pieces but it's not like I can show up to class and I understand what the heck we were doing it's like you know I show up to class, and I'm an attendance grade, you know *chuckles* and it's like, and then sometimes I get bits and pieces of information but it doesn't really [...]

- P1: that's probably that's been the worst of it is trying to figure out how to to remember to keep you know when we're doing all this when we're creating our our classes and our constructors and our methods and our getters and our setters it's like, that's a whole lot of stuff just piling up really quick and then having to jump around because when I switch screens I I don't remember where I'm at so like, if I have like, when I'm in IntelliJ and I have one class up and another class up and I switch that and you know like we were doing with the gardens and the produce, if I switch from produce to garden, I don't even remember what the heck I just saw on produce, and then having to switch back and forth all the time is like I I tend to do a lot more of that trying to do it really fast so I'm not forgetting what I'm seeing and where I'm at.

- P1: I really rely on the book. *chuckles* I I I just I don't know it's it's how I, like, anything off the computer is just a mess you know, I gotta print everything out, and if I can't do that I'm just, I'm a bloody mess. Because I can't even like, copy things from the computer right so eh, I, it's just, it's just really helpful to me to have you know, the the information I need in a hard copy physical form, so that I can look at it and evaluate it and understand it and I can organize it from there

- P1: Really that I haven't just said it eff it and left *laughs* you know, um, I don't know I I I'm definitely much more confident when it comes to uh being able to read some basic code and understand, like I get the kind of like the gist of it it's like it's like being able to read the newspaper in another language, where you're like yeah, I can I can understand, I get the gist of what's going on here, but I may not be able to understand every, I can't, translate every word here, but I I kind of understand what's going on with the code, just like you would if you were learning an actual language and trying to you know, read another country's newspaper.

- P1: [...] that's kind of I guess where I I have those aha moments, because all of a sudden I'm typing in code and I'm like oh, you know what I mean whenever you, like highlighting a variable where it's like oh my God, that's connected up into there, that's connect, okay that makes it, okay that makes it er, you know even something as simple as even your brackets where it's just like oh my God you know it it kind of, the little things, for me, is where it's at, um, just, I just kind of keep going back on kind of just the detail of it, you know, and the depth of which we've gone into when you're realizing that when you're allocating memory with the new command, if you're allocating memory *chuckles* you know, you're storing this in a spot in a memory location *laughs* and it's like "huh, I never would have thought about that" you know, it actually has to store memory, you know, or it actually this you know even like those little dot syntaxes and stuff and what they do, why they're even there you know, instead of like "just do this line of code and it's all you need. Just do this line of code" well, use this line of code because you know, this is how everything works, you know [...]

- P1: You know, you can have access to something but you're missing different pieces and they tend to be important bits and pieces and, you go to try and apply it and you can't do that, and especially when you get it right once or twice, and then you can't get it right again, it's just like, you know it's it's like going to pick up a fork, you know, you don't think about it, you just do it, and then all of a sudden you just can't do it, you know. And it's like, what's up? Where it's like, I don't know if you've ever been paralyzed before but its *chuckles* you wake up and you can't move and you're like what, but that's what it's like, you know you're you're in it one second and the next you're just like, you're lost and confused and you're like well how do I get back? You know, and it it's it's irritating because there's, you're just missing these important bits and pieces of information, and that you weren't missing before

- P1: INTERVIEWER: Do you, do you remember me waving around rubber duckies at all that day? PARTICIPANT: Oh I do remember the rubber duckies. See, and that's that's what's so crazy because it's just like, it's just like a factory you know, you'd go on with your original, and you'd produce so many variants based on the original it's it's that's like it's just an assembly line to me, at least that's what I saw the rubber duckies as you know, like like which, you wanna change the color well you just change, change, you know, go to your setters and change it up from there that's like, but uh, when it came time, when it comes time to actually applying it it's like, I just shut down and it's like everything just goes "boof"

- P1: Like I said it's it's, this whole experience is just amazing really, it's been a night and day difference from my previous experience at Tech you know and I I really appreciate it, and it's been really helpful, it's just, man, the the online thing just killed me it just it it crushed because you know, things that are going on everywhere and in the home and then trying to maintain you know what's going on with school uh you know, what's going on real life, what's going on in your immediate situation it's like that's just, that's, you know especially when you get overwhelmed really easily you know all the problems that are like, they demand you address them now and you just, you're constantly having to, and everything's a priority but you can only, you can only pick one thing to do at a time, and it's like well, that, you you're only gonna be able to do what you can do and that's you know I I try to do that without being disrespectful to you guys because it's like, you guys you guys know it's not like you guys are irritating me or causing me problems or anything else like I said it's just things that are going on with me and I'm trying to figure out ways to deal with and manage, and then you guys are trying to figure out how to help and it's like, you know it's so frustrating because it's like, you get, you got the help that you're looking for, and the help that you need, but you can't even like, you're not even like, not you but like, I'm not even able to accept that help fully or use that help fully and it's really frustrating because it's like, in my, in my, from my point of view it's like you know there's no reason at least that I a normal person should be failing the class in my in my opinion because it's like man it's just it's too easy to go and get the help you need and because the support's there you know, but it's like, with me it's like I I can't even, yeah, I can ask for help all day but it more or less er what it amounts to is I'm just wasting that person's time you know and because that's that's at least how I see it because it's like I can't, how can this person help you if you can't even you can't even absorb and use the information that they're giving you or the techniques and the methods because it's like, it's not, nothing it's nothing they're doing on their end that's the problem it's just what's going on with you, er with me I should say, that is the issue there

Participant 2

- P2: It's just being able to find the right syntax so if I feel stuck on [...] I I find I get stuck on very minute things for significant amount of times that shouldn't take as long as they do

- P2: Probably the most useful thing is being able to bounce ideas off of other people, I think that environment's really great. Also the way you take questions when you can in class and a lot of questions from other students are very uh helpful, a lot of the times I don't know or find myself lacking questions to ask whether I want to or not so

- P2: um, everything that we're doing with working with groups in the lab and doing all these other assignments, it's huge in especially engineering if not what's probably the most common thing in this class is probably the CS majors is just straight up people communicating with each other. So how much I hear from *pause* even where I was working this past summer is, just people being able to communicate with each other.
- P2: Keeping up with the workloads, sometimes I uh put things off for longer than I should but uh, that's a - work in progress, definitely made a lot of progress on that this semester
- P2: [...] so I guess I like to talk about things that are interesting and to learn more about different things. Whereas some professors are a little bit off-putting and somewhat even intimidating in the way they lecture, you're very out there, outgoing and enthusiastic about the way you teach, I think. [...] Well, the ducks are fun. INTERVIEWER: *chuckles* the ducks are fun? PARTICIPANT: I guess to a lot of those interactions, and even what you did in the first day of lecture, that was uh, *pause* definitely one of the most engaging exercises I've ever experienced in a lecture, if not the most. Especially here at Tech.
- P2: INTERVIEWER: When you look at the PDFs do you remember some of, like for example, you were talking about the ducks um, now, that - we've done a lot of examples with the ducks um, but are ever times when you are um thinking something through and the ducks come to mind or things that happened with the ducks come to mind? PARTICIPANT: Uh, some stuff with different constructors usually comes to mind when I think of ducks uh definitely *chuckles* uh *pauses* some stuff with methods too, but I guess it's just general association at this point with programming with ducks now too so
- P2: I don't usually have strong feelings about homework unless they're super personal, so it was pretty good there. The uh, stuff you had us do uh this past weekend for the mentor stuff, that was super personal, I had some definitely strong strong feelings with that so. But that was good.
- P2: Only needed this course just so that I could get a specific credit on my degree. I didn't know what I wanted out of the course, I thought it was going to be fun when I signed up for it anyway. I have definitely gotten more than what I expected out of the course so. It has helped me develop skills that I've wanted to learn and flush out for a long time.

Participant 4

- P4: The peanut butter sandwich thing was funny. INTERVIEWER: *chuckles* it was funny uh, did it, was it impactful for you learning anything? PARTICIPANT: Mhm INTERVIEWER: What do you feel like you learned from that? PARTICIPANT: It's like you can't just make assumptions and the computer knows what you want it to do
- P4: I think I'm probably most proud of that Rubik's cube solver I did. INTERVIEWER: Alright, so so, you uh, you're proud of that, from [PRIOR TO CS1], is there anything that you're proud of from this semester? PARTICIPANT: Um, I brute forced the lab the other week *laughs* where I had to check if a String was equal to another thing by doing a monstrous loop inside a loop inside a loop INTERVIEWER: That sounds terrifying PARTICIPANT: Er

no it was an if statement with like five if statements nested inside each other INTERVIEWER: And you're proud of that thing? PARTICIPANT: Hehe, yeah *laughs* actually I figured it out in the time frame of the lab and just did it so. INTERVIEWER: Do you think it was a good way to do it? PARTICIPANT: *laughs* it was a way to do it within the time constraints given

- P4: Probably would have made an object called like CubeTurner or something had a constructor that could be put in some parameters to have it act on the different sides, and then it instead of having ten bazillion lines of code, which I did in the Rubix Cube solver it was like 1,700 lines of code, um, instead of doing that I could have just had one constructor which constructed a bunch of cube turners that would all act on different sides.
- P4: No, the ducks were helpful just like anything that takes and makes like a physical object of something helps otherwise it's like abstract
- P4: Um because it matched the template and I think I may have plugged something into IntelliJ too to check it

Participant 5

- P5: Um. Like going through uh the I I think that you do a pretty good job of like uh creating analogies to kind of describe like uh some of the uh concepts and I I think that's helpful, especially for people who are first learning things. Like if you're just like alright, here's this, here's what it does, versus here's what it is, now let's relate it to something else that you can kind of uh like make sense out of. INTERVIEWER: So that's uh, that's interesting that you said um, analogies, can, do you have an example of one or two of those that I've like, even if you don't remember the exact analogy, can you remember any words or things that were related to? PARTICIPANT: Er uh I was just kind of thinking of like um all the stuff that you've done like ducks and like in your PowerPoints you'll just see random stuff like that.
- P5: Mindset of, we have a lot of work to do and we're just going and kind of go through and get it done so it I've definitely fallen into that uh category quite a bit just plowing through assignments and stuff you know like alright we're just going to get this done without paying like too much attention to like what's going on beyond that.
- P5: Uh I'd go to lecture more *chuckles* INTERVIEWER: *laughs* Don't think you have to say that just because you're talking to me, by the way. PARTICIPANT: Oh, no *laughs* that's not why *laughs* INTERVIEWER: So why do you feel that that would have been more impactful? *laughs* PARTICIPANT: Uh, just because like uh I would just skip class somedays like if if I definitely found towards the beginning of the semester I was quite busy and uh procrastinating a little bit on my homework, so I just like skipped a couple classes uh to get my homework done that would be due later that day just so I had enough time to do it, because I just like planning ahead, and then when I'd go to do uh like the this class's homework, then I'd be like oh, I missed this week's both of the lectures this week, and I don't know exactly what's going on with this, so then I would have to spend even more time like going through the uh lecture slides and trying to teach myself.
- P5: Uh I found that it made sense, it seemed like uh what we learned just kind of built off each other, and it like just it seemed like uh just more of a natural progression and so just jumping around in the textbook didn't really bother me that much.

- P5: Uh I would like to. I don't have any concrete plans but I mean, even uh I find that I want to like design or create like a couple of apps just for like little personal things like organizing files and stuff, and that would be useful just for through my downloads folder especially, that's just filled with random stuff *chuckles*
- P5: Both INTERVIEWER: Okay, so both of those instances, even though the number in the square bracket meant something different in both of those it, it made sense how it was used? PARTICIPANT: Yeah yeah, and I don't really have any trouble discerning like the two different uses I'm like, when you first define it this is just the size and then when you go to use it, this is like the element and the numbers are different.
- P5: Uh like my responses, some of these answers are just very vague where I feel like if I were to put the answer and then like an explanation or something that that would just help gather like understandings eh as your thinking of like what's going on like as you're doing it, because like if you have to explain it then that in itself requires a certain level or a certain ability to know what's going on so it just uh probably having like one to a couple word answers probably aren't the best thing but

Participant 6

- P6: [...] but I always kind of try to figure out like what's going on like with the hardware aspect um of it uh of the whole you know, what, what you're actually doing and like memory allocation, stuff like that, so I definitely take more of a hardware approach than um, probably like computer science majors or something
- P6: I don't really know what's out there exactly yet, I'm just kind of learning as I go but one thing I'm definitely looking forward to learning is kind of more of like the practical um applications of uh objects kind of uh because you know we learn I think like the latest example was kind of like the you know the produce garden bunny thing, I think those were like the big three classes that we made and um like it it makes sense kind of, you know, how we use those like for that situation how we'd use you know, how everything was kind of set up but it'd be cool to learn how you know in real life in industry what you'd actually kind of do, because you'd never really make like a produce bunny object you know in in industry I don't think, so it'd be cool to kind of learn some like real life applications and how that stuff works
- P6: Um I think kind of like when you sort of uh relate it to something in real life you know, the I know I just totally ripped on the garden produce bunny thing but that uh it it really does like kind of when you relate it to something in real life like that it does, I think help me you know kind of grasp like the big overarching concepts of it um and like I remember the hockey rink thing, like the hockey you know, the hockey rink and then there's the player and the ice or whatever and that, I remember that, that one really stuck with me uh just kind of relating stuff to real life [...]
- P6: I guess, um, I didn't really like how you can't really what is it, you can't really it's like changing the size of it after you declare an array which is something that was sort of you know, kind of a weird um because you can usually change all of them on the fly and you know in all the other stuff that I've used um, I guess that was kind of weird and declare, it's *chuckles* I guess one, it seems like when we were learning arrays it's it's like Java's not really meant for arrays as kind of some of these these other languages are, that was definitely one feeling

uh um, but yeah I think ArrayLists are a lot more easy to use still kind of weird but um a lot more easy to use than those like primitive array things that you couldn't really change the size at all.

Participant 8

- P8: I think uh that it's a very creative work um because I have to uh connect my idea to the specific codes um and uh I I always think that's a kind of dramatic *chuckles* I uh, magic
- P8: the interactive objects uh covered uh a few weeks ago I think mmm, it's like a system of many um many animals uh or humans uh that can do the job together *chuckles*
- P8: Uh I think this the uh rabbit one, uh and uh also the duck one because we need to uh, fetch constructors uh setters, getters, and toString method mmmm because I I I like animals *chuckles*
- P8: Yeah, I'm I always remember the several ducks *chuckles* uh because we need to um spend uh the alias uh the concept of it, also we should uh, create several constructors uh for different ducks mmm it's quite helpful uh to use tools on, I mean this uh, ducks are quite cute *chuckles*
- P8: I think that's the objects, uh object, and the and its methods, because mm, I feel confused uh when I was first um coded it, I should use object a class method, because that's the noun does verb, and that format is should, is um, correct, and I should use it. Mmm, because I thought uh, it should be the class name, um class then method *chuckles* mmm, but uh, when I read the textbook, I start to know uh I should use the object name * chuckles*

Participant 9

- P9: I think objects because it was just like something that I didn't know about before. And I think, I don't know something that like once you understand I feel like it makes all of programming more understandable.
- P9: Mmm, I think because I'm more confident with it for one, for two I like Java a lot better than Matlab um, but I don't know, I guess before I just had like negative connotations because all I could think of was how hard it was when I didn't know what I was doing *chuckles*
- P9: I'll be honest, I didn't look at the syllabus, so I didn't really have any expectations *laughs*
- P9: Um, I really liked the ones where we made like the conservatory er like because I don't know it felt like we actually like made something, that one and then the last one, the last project that we did, I can't remember what it was called INTERVIEWER: The text predictor? PARTICIPANT: Text predictor, because like it actually made something that like you could use so it felt like I don't know, I I did something *chuckles*
- P9: I think that like not just showing us like on the program was really helpful like and I don't know like for objects too like having like us actually make like TRexes instead of like automatically going to like an actual like programming assignment we would have it like made more understandable by like relating it to like the world somehow, and like, or like the rubber

duckies too, like naming them, it made it so much easier to like, understand how they're supposed to be interconnected, because like, those things you already know how they are connected, if that makes sense.

- P9: Yeah, like the rubber duckies, I even liked the whiteboards just because you got to like practice writing it exactly how it should be, like we had talked about it, how it would be in the real world, and then it was like okay now put this in like a code which I liked that, it made me understand like, instead of just knowing like what characters to type in the computer like it made me understand why I was typing em and it made it like a lot easier to apply it
- P9: Between like static and dynamic and programming, because I think like, I would never like, if I open like a different like, program like Matlab and I didn't know the difference I'd feel like it would be really hard to figure out how to code in that language, but if you do you'd have an idea of what kinds of things you would need to type in, and be able to figure it out.
- P9: I think just the fact that like, I feel confident that I can write a program somebody asked me without like, needing help, I don't know, I had never felt like super confident with programming but I definitely if somebody like asked me to write a program in Java I'd feel like I could, which makes me happy. *chuckles*
- P9: Yeah, I had actually like pretty low expectations because I knew it was like a really beginner level class and it like way surpassed expectations *chuckles* INTERVIEWER: *chuckles* why is that? PARTICIPANT: Um, I don't know, because like, I literally was only taking it for the fact that I needed it as a prereq for my minor, and so I was like, well I'm just gonna like do this class and it was like gonna be one of my side classes that's just kind of on the backburner, I'd just try to like do alright in it, but then I like really liked it, so I would always do that homework first and then like, I don't know, I learned a lot more in that class than probably in any of my other classes. [...] Now I'm kind of thinking about taking some more classes maybe getting a masters in computer science [...]
- P9: I used the slides a lot when I was working on the homework or like on the quizzes I really did not like the quizzes at all. INTERVIEWER: Like just in general or that week? PARTICIPANT: In general, it was just kind of like, it was it felt like a lot of busy work, like those didn't really help me learn that much I don't think, I think like the exercises sucked but like, I did learn from them like the concepts better and the programming I really liked, the quizzes I literally would just like, I'll be completely honest, I would go through, take it, figure out which ones I got wrong, and just take it again until I got like enough points and like I was not engaged for those at all.
- P9: Whenever you ask a question like that, I'm like oh it's probably like because it prints memory uh I like wasn't positive but I was pretty sure that you wouldn't ask that question unless it printed something other than what you would expect.
- P9: I don't know why, but like just using the word instance variable like, right off the bat I was like oh man, this is where we get into language that I don't know like that's not *chuckles* I kind of started freaking out and then um I think when you like related it like to like I don't know if you named what, what were the ducks names? INTERVIEWER: We had Jonathon, and Quackington? I don't know PARTICIPANT: Yeah so when you *chuckles* so if you're like oh Jonathon is like an instance of like the Duck class like that would make sense, because there's like there can be like all kinds of different ducks, but like, Jonathon's one of them so like, when you described it that way, like it made sense. Which was like, also kind of an

aha moment, that's when like, I think I understood what like an instance means, and then I don't know, I didn't like the whole like, private variables thing like it annoyed me that we couldn't just make them public because then you have to make getters and setters and I didn't understand why which I do now I think but that was like really frustrating to me. I just wanted to make them all public and stop like, messing with all the other stuff *chuckles*

- P9: Yeah, just because like, I already like understand how those are like related so it helped me understand like how they were supposed to be related in my program. And like if you had had like, like if say for example like we were talking about machines and you were just like giving me names of that stuff like I would have been like what is happening like I would not have understood like because I know the Duck names are like Ducks and they all swim in the pond like I already know that in my head, like it made more sense *chuckles*
- P9: Yeah, because people like, everybody has like different styles too or like have done it for a different application so like, they have all kinds of other code in there and like, I'm not one to go to copy and paste from like Stack Overflow but like, I will look through it and look for examples of code that like might help me with mine, and a lot, it's really hard to find like, good stuff that's actually helpful.
- P9: I would say that only really happened in like the first lab after we learned it, and after I kind of understood it, I would get really excited because like, I don't know, it was something new that I had just learned that like I figured out how to use by myself, and then it was like really rewarding to get it to work, especially like the I don't know like, the conservatory and like some of the ones that were like, bigger and you made like a whole scene, was really fun and you're like oh look, like, I have my Duck pond or like, I don't know, you like made your own little thing kind of like how people like those video games like Minecraft where you make your own stuff, it was kind of like that. It's like rewarding when you get it. INTERVIEWER: When you're kind of building your own little world almost - it's it's rewarding, because you get to see it modeled. PARTICIPANT: Mhm, yeah, yeah.

J.2 After CS1 - Student Perspectives

Participant 3

- P3: Honestly that bunny program like just *chuckles* there's just something kind of fun and you know, it was a more interesting than if I was *pause* makin' files open or something weird. [...] INTERVIEWER: Do you think it made it make sense too, or? PARTICIPANT: Cause everything was like, there's a, you already knew how they like connected, so you kind of already had that intuition of what needed to go together so now you had to just had to bring the computer to it, which made thinking through the logical part having to focus on the syntax
- P3: Oh sorry um, *chuckles* it was interesting I think just because like, it was a really good way to like learn in time like, I felt like I was playing with my own little zoo on my computer and it was kind of fun *laughs*
- P3: Like everybody else seemed to be getting it, like it, like you had to write everything down or I can't figure out like how these steps chain, so I think maybe in some ways it was more,

it was difficult because it was just a harder concept and then with your classmates kind of understanding it and like being able to think through the loops a lot faster in their heads, it's frustrating

- P3: Demonstrations like, those were really helpful, and the way that you like used like analogies or linked things to other things that like, weren't programming related was nice. INTERVIEWER: What did you think was nice about that - sorry. PARTICIPANT: It just like, something ab, something that I didn't know before but having something I kind of did understand between the two a little easier. The slides were nice, just well laid out, I do really love like slides that I can go back on and look at, so that's sort of something that's like it's just really really vague that just never really works for me INTERVIEWER: So you sort of felt like the slides had enough content to be helpful? PARTICIPANT: Go back and like, read em and laugh. I mean I still go back and look at em. INTERVIEWER: Really? PARTICIPANT: To get my assignments, yeah, I forgot to, I forgot to download the files but [FRIEND NAME]'s got em on her computer so she emails them to me when I need something

- P3: I think of your duck, your week 8 ducks. Like that's one of the first things that comes to my mind INTERVIEWER: *chuckles* the week 8 ducks, what about them comes to mind? PARTICIPANT: Just the fact that it was a really like, that's kind of, was a really good way for me to start to understand objects. We started more at like, I mean, instead of starting just jumping into the code, it was a little bit more intuitive a little like, this is how it was like actually like real world, now you just have to like, starting the idea of the logic before the computer came in was nice.

- P3: And then exceptions and you throwing erasers at me INTERVIEWER: *laughs* PARTICIPANT: I'll never forget that *laughs* INTERVIEWER: *laughs* oh noooo PARTICIPANT: No it really helps though! Like when I got to files and I needed to think about whether I needed to throw or catch and then I ended up doing both, it really did help *chuckles* INTERVIEWER: So you actually did end up throwing and catching? And trying to like walk that through with the erasers? PARTICIPANT: In 1122 INTERVIEWER: You did in 1122? PARTICIPANT: Where, where I caught and threw an exception, yeah. INTERVIEWER: And then, you you thought about it based on that example? PARTICIPANT: Yeah I was looking at it and I was like wait, what's that, and then I was like okay this isn't gonna work *chuckles* it really did make a difference.

- P3: I'd definitely say when I signed up my desire was all was pass a needed prereq, I don't think I saw the importance of it, until I started sitting in lectures and actually learning. At first like, what I wanted to get out of it was a lot more of the knowledge and understanding than I did at the beginning.

- P3: We did objects later than the book did, didn't we? Er earlier, we did em earlier. That was helpful, because like once you kind of understand objects, other things start to work. And then you realize that everything is kind of just this object and then you can kind of know how to interact with it better, that was cool.

- P3: Right after we had learned objects and then like, I had that huge aha moment when you said Scanner is an object, like Scanner was a, like that all of a sudden just weirdly clicked, it was this huge lightbulb moment.

Participant 7

- P7: Uh yeah definitely I definitely have a lot more respect for it because it's it's an art like before I I didn't think of it like that but it's definitely an art and I have mad respect for people who can nail the programs because it is so hard um yeah lots of thinking, it's like it's more creative than I thought, I thought it would be super straight forward but it definitely does take some creativity

- P7: It was making what were we making? Like monsters or something, I don't really remember it was one of those programs with like making objects when we first getting into it, I think that was that's when everything came together for me, and that was like my favorite part I don't know, that's just when programming started to make sense as like programming, it wasn't like, I don't know if that makes sense but yeah, that's just when it hit me *chuckles* that's my favorite section.

- P7: Just honestly, December as a whole, was really *chuckles* kind of kicked my butt, I think, it might have just been it was end of semester but, I I definitely started to struggle with the uh, that home stretch there. I don't even remember what the topics were, I just remember being lost *chuckles* [...] Oh for sure, yeah, I didn't do anything over break, like I my brain was dead when I came back and just with all the other classes getting so busy it was just like awh. It's hard to balance it all but that's not specific to like the course really, it's my life *chuckles*

- P7: Um, the peanut butter and jelly thing, that was fun. I didn't like, I never, I just struggled with using the book. I didn't it's just so much it's like overwhelming so just, everything we did in class um. When you'd program on the computer and like print it back too, that was helpful because like I'm someone like I understand concepts pretty well but syntax I mean, I still struggle with syntax so much like I just, I know what I have to do, I don't remember what I have to type, so doing that is really helpful for me. [...] Did you throw stuff? I don't know if that was you. [...] INTERVIEWER: Do you remember ducks at all? PARTICIPANT: Yeah that was the second thing I said, I don't know if it cut out but

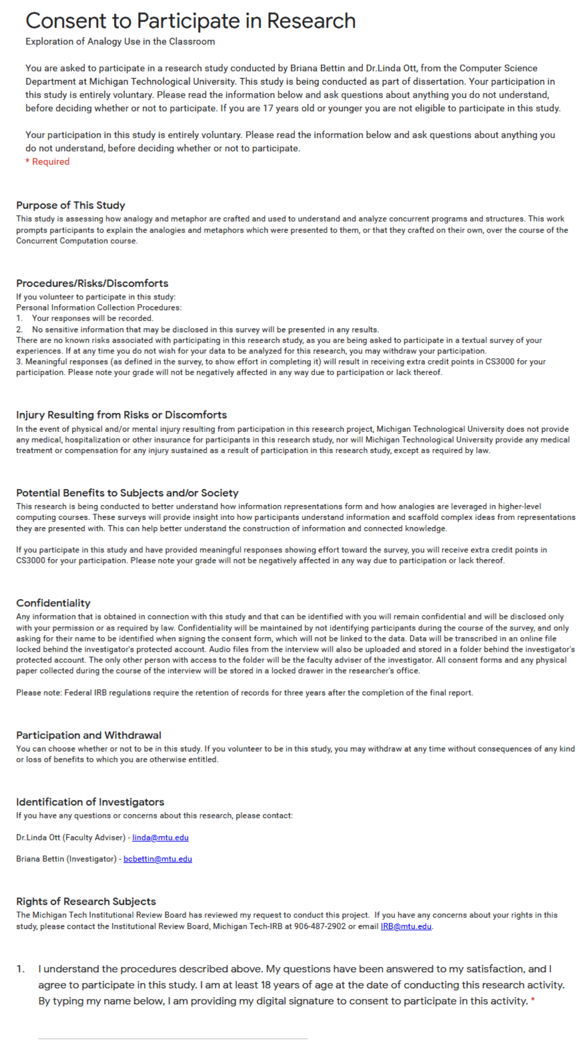
- P7: Like what programming actually is. Because it's not *chuckles* what I thought it was. I think just understanding that it takes a lot more than just like, typing words, like you have to think about it really critically and I don't know, it's yeah. It's like learning a language, like French, but you have to be creative with it, I don't know, it's just wild *chuckles* That was my biggest takeaway

- P7: Uh I would have done the whole month of December differently, I would have tried a lot harder because there definitely was a gap then when I got to 1122 that I was like shoot like, I wish I could have just been stronger that month and like *chuckles* I probably would have been a little better off now but, I don't know, just keep pushing through like it's something that you can't really give up on, because it it doesn't stop. Like you just keep going, so you kind of have to keep up with it, and I know that's how it is in industry too, like you're always learning, so. There's no time for for rest *chuckles*

- P7: Yeah definitely, I went into 1122 kind of discouraged but that was kind of like my own thing because I slacked but um. Kind of made me scared for 1122 I was like shoot like am I way behind? I don't know. But again that was just my own *chuckles* my own issues.

- P7: I'm definitely nervous for the next course um because it's like same thing, I mean this semester is definitely more challenging learning from home but like, ahhh I just the burnout again like I just, I get nervous for the next step because I'm like, if you don't like, ahh, I just like a whole summer without it you know constantly being there I think will be hard, I think you definitely have to kind of brush up your own skills on the time in between but I don't know, that's just how I feel a little discouraged right now too. But, it was the same thing last semester so I know I'll get through.
- P7: It was after it was after we created like, oh my gosh it was um, bunnies and gardens and it you, what did you have to do? *pause* you would put like, fruits and vegetables in it, and what was that? INTERVIEWER: Uh, that was the lab. PARTICIPANT: That was a lab? Okay, I I remember that lab. We were just like, I uh, and the gardens? I don't even remember what that was about. That was a big that definitely helped me with the programs that came after that, or the program that I did before that helped me with the lab, one way or, I don't know. I just remember understanding that. Not understanding it, like at the beginning of class, and then my lab partner was like no look, and then I was like woah. And I don't remember exactly what it was but.
- P7: It fit together nicely like once you understood what you were doing if that makes sense like. And that was the same thing with the other program that I'm obsessed with, like the creatures, like everything just kind of like it just has this, you can feel it, like when it makes sense, I don't know what it was, I don't even remember the prog, what it was about, but, yeah.
- P7: No I think I I had pretty low expectations for myself *chuckles* going into it uh and I don't know, I think I surprised myself with how much I was able to accomplish like, on my own and actually understand it so, I exceeded my expectations *chuckles*

Concurrent Analogies Survey



Consent to Participate in Research

Exploration of Analogy Use in the Classroom

You are asked to participate in a research study conducted by Briana Bettin and Dr.Linda Ott, from the Computer Science Department at Michigan Technological University. This study is being conducted as part of dissertation. Your participation in this study is entirely voluntary. Please read the information below and ask questions about anything you do not understand, before deciding whether or not to participate. If you are 17 years old or younger you are not eligible to participate in this study.

Your participation in this study is entirely voluntary. Please read the information below and ask questions about anything you do not understand, before deciding whether or not to participate.

* Required

Purpose of This Study

This study is assessing how analogy and metaphor are crafted and used to understand and analyze concurrent programs and structures. This work prompts participants to explain the analogies and metaphors which were presented to them, or that they crafted on their own, over the course of the Concurrent Computation course.

Procedures/Risks/Discomforts

If you volunteer to participate in this study:
Personal Information Collection Procedures:
1. Your responses will be recorded.
2. No sensitive information that may be disclosed in this survey will be presented in any results.

There are no known risks associated with participating in this research study, as you are being asked to participate in a textual survey of your experiences. If at any time you do not wish for your data to be analyzed for this research, you may withdraw your participation.
3. Meaningful responses (as defined in the survey, to show effort in completing it) will result in receiving extra credit points in CS3000 for your participation. Please note your grade will not be negatively affected in any way due to participation or lack thereof.

Injury Resulting from Risks or Discomforts

In the event of physical and/or mental injury resulting from participation in this research project, Michigan Technological University does not provide any medical, hospitalization or other insurance for participants in this research study, nor will Michigan Technological University provide any medical treatment or compensation for any injury sustained as a result of participation in this research study, except as required by law.

Potential Benefits to Subjects and/or Society

This research is being conducted to better understand how information representations form and how analogies are leveraged in higher-level computing courses. These surveys will provide insight into how participants understand information and scaffold complex ideas from representations they are presented with. This can help better understand the construction of information and connected knowledge.

If you participate in this study and have provided meaningful responses showing effort toward the survey, you will receive extra credit points in CS3000 for your participation. Please note your grade will not be negatively affected in any way due to participation or lack thereof.

Confidentiality

Any information that is obtained in connection with this study and that can be identified with you will remain confidential and will be disclosed only with your permission or as required by law. Confidentiality will be maintained by not identifying participants during the course of the survey, and only asking for their name to be identified when signing the consent form, which will not be linked to the data. Data will be transcribed in an online file locked behind the investigator's protected account. Audio files from the interview will also be uploaded and stored in a folder behind the investigator's protected account. The only other person with access to the folder will be the faculty adviser of the investigator. All consent forms and any physical paper collected during the course of the interview will be stored in a locked drawer in the researcher's office.

Please note: Federal IRB regulations require the retention of records for three years after the completion of the final report.

Participation and Withdrawal

You can choose whether or not to be in this study. If you volunteer to be in this study, you may withdraw at any time without consequences of any kind or loss of benefits to which you are otherwise entitled.

Identification of Investigators

If you have any questions or concerns about this research, please contact:

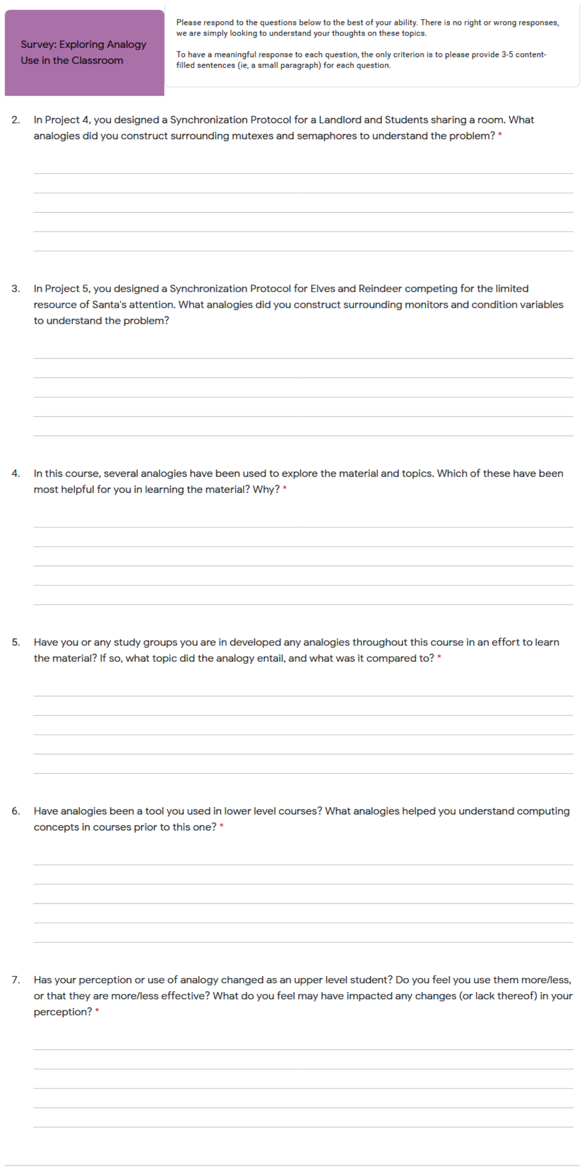
Dr.Linda Ott (Faculty Adviser) - linda@mtu.edu

Briana Bettin (Investigator) - bcbettin@mtu.edu

Rights of Research Subjects

The Michigan Tech Institutional Review Board has reviewed my request to conduct this project. If you have any concerns about your rights in this study, please contact the Institutional Review Board, Michigan Tech-IRB at 906-487-2902 or email IRB@mtu.edu.

1. I understand the procedures described above. My questions have been answered to my satisfaction, and I agree to participate in this study. I am at least 18 years of age at the date of conducting this research activity. By typing my name below, I am providing my digital signature to consent to participate in this activity. *



Survey: Exploring Analogy Use in the Classroom

Please respond to the questions below to the best of your ability. There is no right or wrong responses, we are simply looking to understand your thoughts on these topics.

To have a meaningful response to each question, the only criterion is to please provide 3-5 content-filled sentences (ie, a small paragraph) for each question.

2. In Project 4, you designed a Synchronization Protocol for a Landlord and Students sharing a room. What analogies did you construct surrounding mutexes and semaphores to understand the problem? *

3. In Project 5, you designed a Synchronization Protocol for Elves and Reindeer competing for the limited resource of Santa's attention. What analogies did you construct surrounding monitors and condition variables to understand the problem?

4. In this course, several analogies have been used to explore the material and topics. Which of these have been most helpful for you in learning the material? Why? *

5. Have you or any study groups you are in developed any analogies throughout this course in an effort to learn the material? If so, what topic did the analogy entail, and what was it compared to? *

6. Have analogies been a tool you used in lower level courses? What analogies helped you understand computing concepts in courses prior to this one? *

7. Has your perception or use of analogy changed as an upper level student? Do you feel you use them more/less, or that they are more/less effective? What do you feel may have impacted any changes (or lack thereof) in your perception? *

Appendix L

Concurrent Analogy Responses

> Has your perception or use of analogy changed as an upper-level student? Do you feel you use them more/less, or that they are more/less effective? What do you feel may have impacted any changes (or lack thereof) in your perception?

Following are the responses provided:

- As my understanding of programming deepens, I am beginning to use analogies more, because thinking programatically is beginning to become second nature to me, so I no longer have to think about the exact terms I should use for a problem, and can better understand things by looking at real life to relate to a problem in programming.
- Analogies can be useful for understanding a topic, but when they are used to describe specifications for an assignment it can be hard. I think that the program specifications for this course that revolve around something mathematical like sorting or prime finding were much eaiser to understand. The assignments based on analogy leave more room to mis-understand the purpose and intent of the assignment. Analogy is a useful tool for learning, but it is important that you come to understand the topics covered in a way that isn't directly reliant on analogy, because often times analogies don't match perfectly, and that can lead to confusion.
- I think they are more useful as an upper-level student. Lower level classes have simpler ideas, and often times issues are in the technical detail, not the big ideas. They are much more effective in 3000 and 4000 level classes in my view.
- I use analogies more than I did before, but I think that's because I can relate to topics I understand, like for loops.
- The most extensive use of analogies I have encountered is in this course, Concurrent Programming. After reviewing these analogies, I definitely believe that they have positive effects on improving understanding. While it may be expected that an upper-level student be able to understand the raw concepts as they are presented, I do no believe that their omission from instruction would be beneficial. Providing several avenues through which students can approach a problem increases the ability of any individual to approach the problem. I am unsure if I could come up with analogies for concepts which I do not understand, as I would be unsure if it is an accurate analogy. However, an analogy presented by or verified by someone who you trust understands the concept well can help you grasp the concepts more easily.
- I would say that I use analogies far less now that I am an upper-level student. However, I do not think that they are any less powerful. I think that analogies help to greatly clarify understanding between people. When explaining a new concept to someone, an analogy will

help them understand many aspects of that phenomena without explicitly saying it. I would say that the reason I think they are less frequently used for me is because I have a greater understanding for many technical aspects of computer science. Therefore, I do not have to use an analogy to understand how something works. These analogies would normally be related to other technical examples or constructs versus an analogy as a comparison with the outside world. With that being said, I would say that most of the analogies I use now are relating one language to another language or other very technical analogies.

- Oh I definitely use them more. A whole lot more. I have to. I mean, I don't see another way of doing things. I really don't. For me, analogies are all but mandatory for understanding code. If I run across a piece of code that I cannot break down into some kind of analogy then I'm probably not going to be able to understand it. I need to be able to physically see something happening in front of me (in my head) in order to properly code it myself. Concurrent Programming in particular has really forced me to solidify this process. It has given me a huge appreciation for applications that are not threaded. I don't think I'll ever again look at some complicated piece of code that has only one thread and think "how could this get any worse?". Because now I know that it absolutely could. It could be a lot worse! That and just doing it. I cannot over-stress the importance of simply coding A LOT in order to become a better programmer. It's probably the single-most important thing you need to do, and I don't think a lot of students realize this. Even I don't code nearly as much as I should. That's why I think programming assignments are so incredibly important. Numerous times this semester (and previous semesters) I was totally lost on a given subject. Just completely lost. Then I'd have to write a lengthy program using that subject for hours and hours. I'd lose sleep and be coding for what felt like days, sending help-emails to professors left and right.. Then after what felt like an eternity and 28317 bugs, I'd complete the program and finally understand it. All the slide-reading and studying in the world cannot replace simply being forced to use the code you're learning about.

- I don't think my perception has changed. If I was better able to come up with my own analogies, I think it would help to better understand the material for a class. If you can make an analogy for a problem, then you must understand the material enough to do so. I feel I use them about the same amount as I used to and the effectiveness of the analogy depends on how well I understood it originally. What impacts my perception of analogies is I have a hard time understanding them and making the comparisons between what I am trying to do and the example that was given in class. I have a really hard time understanding word problems and I need to see many examples of a math problem before I can do it on my own. I need to be able to find the right analogy before I can make it useful to me, like the dining philosophers problem which is easy to remember.

- Absolutely; as Ive gone through upper level courses, my use of analogy has increased a ton. Ive come to perceive it as an integral tool to understanding advanced concepts and mastering them. Analogies are so incredibly effective now compared to lower level CS courses; thats not to diminish their potential usefulness at any level, but as concepts become more and more advanced, relating them to analogies becomes insanely useful. This increase in the difficulty of concepts is what caused me to perceive analogies as much more useful over the years.

- I think the use of analogies is extremely helpful in the teaching of Computer Science concepts from the beginning to the end. I find myself looking for more analogies as the concepts get more difficult as they provide an easy, concrete explanation for some difficult, abstract concepts.

- My use of analogy hasn't changed much as an upper level student, I typically try to abstract away chunk of code in a program at a time to make it easier to describe the program as a whole, as well as help with debugging. I wouldn't say I use them more or less, but as I get more adept with programming concepts my analogies get more accurate and easier to describe, so I would say my use of analogy has become more effective. I think most professors don't have effective ways to simply describe a difficult concept and that harms students understanding overall, which is part of why I have slowly improved at describing difficult concepts to other people once I can grasp them myself.

- I feel I tend to use analogies more self-consciously and maybe less overall as an upper-level student. I also find I'm more inclined to make analogies to some other computer science concept I learned earlier rather than a concrete situation from the real world. I believe analogies have about the same overall effectiveness for me as when I was a freshman. One thing I have noticed is that slightly inaccurate analogies can interfere more with my getting the concepts as an upper-level student in a way that wasn't an issue with 1000 and 2000 level classes. Things that impacted my changes in perception are having more baseline knowledge about computer science (new topics often look familiar from another context), finding analogies I generate during class to sometimes cause me to miss a crucial 30 seconds of lecture, and having formal training in using analogies as a lab assistant.

- I think that my perception of analogies as an upper-level student has improved as using analogies has been able to help me take a different view on topics and enhance my learning. I have been using them more and they have been effective with understanding more complex topics. I feel that as some of the content in the classes has been what changed my perception as much of it has been more difficult to grasp compared to previous courses.

- My perception of analogy has changed as an upper-level student. I feel that it is because throughout my courses as concepts became more complex I reached a point. Where it was easier to understand by comparing rather then examining the definition at face value. They became more effective for me as the concepts went farther away from my typical thinking process.

- I'm definitely using analogies more as an upper-level student than I used to. Part of that is probably that we're working with more advanced concepts, which makes analogies more useful since it can be hard to describe how these concepts work without comparing them to something else. This change in the difficulty of the material also makes analogies useful even in the case where I could easily describe the concept, since the analogy would most likely be easier for the other person to understand.

- I think that for this course in particular, analogies have been very useful to me. I have basically always though of the mutex's and semaphores as waiting in line at an event and how they are signaled to allow people in. Personally I have gotten through most of my courses without using things as analogies, but rather just working to understand a concept as it is presented to me. I definitely think that analogies are an exceptionally useful teaching tool because it allows more students to be able to understand something if you relate it to something that they already know.

- I definitely use them less. If I was asked to answer this before starting school I would of though the answer would be the other way around. Harder concepts require more methods to grasp. however as I have grown as a CS student I have found that computing concepts in themselves are very comfortable to me and in fact now when trying to learn other skills I use CS analogies to help me. the early years were harder because I was less familiar with how computers worked so I needed more tools to help me.

- My perception of the use of an analogy has definitely changed as an upper level student. I have begun using analogies to describe many of the things I understand in computing to others. This not only helps me further strengthen my understanding of computing topics by thinking about them in other terms, but it allows the person I'm describing it to to understand it. I believe that a deeper understanding of how the computer as an entire system works together has helped change my perception and use of analogies when describing it. Having classes such as Computer Organization and Programming Languages helps to show how the lower levels of the system work while classes like Data Structures and Concurrent Computing explain the higher level uses and effects of these things. It's truly fascinating to be able to understand the complex system as a whole rather than just having a perception based on and understanding of the system from the top.

- I have always like using analogies to explain things and make things easier to understand. As I have learned more and gone through more schooling I find that analogies have become more useful tools than ever. I find myself using them more and becoming more effective. Having to solve more complex problems that have more moving parts have made them more appealing and more useful to have.

- I use them less now, and I feel that they are less effective. However, I have never directly used analogies in my learning since they don't click for me. I do like to see applications of a topic, and analogies are a fine stand in for that, but I would like to see more practical applications of this instead. A component that may have influenced this would be that I started doing enterprise and I had been participating in research, so I started working in more "real world" scenarios, where analogies don't directly help with understanding topics and more direct application is helpful.

- I think my use of analogy has changed a lot as an upper-level student. As a lower-level student (and in this class), I used analogies of real-life objects and such to describe conceptualize computing concepts. But as I've gained more experience with computer science, I think of new concepts in terms of other, more familiar computing concepts to understand them. Especially when learning new languages, it's been very helpful to think of new syntax in terms of C or Java syntax that I'm more familiar with.

- I feel as if when you are a lower-level student, analogies help by simplifying something very new and high level at the time, to something understandable that you can relate to. I think it is used more in lower-level classes to get people thinking about these new topics. In upper-level classes, analogies are still used, and I find them just as effective. I don't think I have to lean on analogies as much to understand topics anymore, but I do appreciate them for them for example problems. It seems like the more of a baseline of knowledge you have, the less analogies are needed. At the end of the day, I still like and appreciate analogies for programs to have a deeper understanding of the topic.

- I will probably use analogies more often. When I get a good analogy, it really helps with my understanding of a topic. Analogies can be very effective. Using analogies and being introduced to more of them has lead to better understanding and perception of analogies.

- As a upper-level student I find I use analogies more than I did in lower-level courses. I feel that analogies are more effective now as they halp me visualize more complex concepts where before I could just memorize the syntax of statements and understand how it functioned. The data structures class definitely influenced my use of analogies in trying to comprehend the concepts in courses as I struggled with the classes material until a friend helping me study explained a solution using an analogy.

- I feel that analogies have remained somewhat similar throughout my classes. I feel that in every class, important new topics usually have some analogy attached to them. The only changes is that the metaphors/analogies often get more complex and can become a bit more twisted and harder to understand as the years get on, which I mainly attribute to having more complex and difficult topics as the classes get harder through school.

- I think analogies are still very important in the upper level classes. They help me gain understanding of how different things work much better than strict definitions, and I find myself constantly asking [INSTRUCTOR] questions in terms of analogies, and when he answers back with them they clear things up for me really well.

- I think that I have been using them consistently throughout my college career. They are always useful to understand topics especially difficult/confusing ones such as concurrency. Most people are visual learners so having an analogy to compare with from something they can picture that they already know is a very useful and powerful learning tool. Any course and course level should use them.

- I feel like I use them more in an upper level courses. I think it is because the concepts are harder so using the metaphors help simplify things. In the very low level CS courses, the problems weren't difficult to understand, so they didn't need metaphors. This would be sorting algorithms, sorting is sorting. Also, data structures were themselves too. Lower level classes had more of the fundamental stuff that couldn't really be transformed using metaphors.

- I feel like analogies are effective for both upper and lower level students. They're useful for lower level students because they're just learning the basics and don't have much of a background in computing. Analogies help tie the concepts they're learning to ideas they can better understand. Analogies can be useful for upper-level students, but I don't think it's as helpful as it is for lower-level students. Upper-level students have an understanding of computing, so it's easier for them to understand new concepts without analogies.

- Analogies have become more useful in upper level courses, and I think that I have started to use them more. This is mostly due to the increased importance of understanding concepts, as well as the increased general complexity of these concepts. I never had to use analogies in the past, but in classes such as concurrent, I can see how useful they can be.

- Sometimes the analogies provided can be too complex for the topic being discussed. I think generally they are effective if explained well. I often find going through multiple examples being more helpful than applying a single analogy to a problem, as it is hard to contextualize it when applied to other concepts.

- I'm not good at it but they do help me a little bit. I kind of wish analogies were used a bit better throughout this school's curriculum because I'm starting to see the value of them just sitting here thinking about it. Not just in Concurrent but in general. My background before coming to Tech wasn't really rooted in computer's so having analogies to tie computer science concepts to other ones would be much more helpful.

- As an upper-level student I feel like I'm taught analogies more than I come up with them on my own. That being said, I have noticed that I use analogies more in explaining something to a peer or younger student (as a [UNDERGRAD MENTOR ROLE]). I feel like the use of analogies for myself is more effective on harder to understand concepts, like those taught in AI and Concurrent. As such, I definitely feel like I use them more now than I did before.
- I found that they are generally useful. the good part is that I would naturally use these analogies when discussing the concepts with my peers, which is good.
- My perception and use of analogies has not changed as an upper-level student. Whenever you're learning something new, an analogy to something you already understand always helps you learn the material faster. I feel that analogies are always effective however, as a student that is graduating I feel I have accumulated enough knowledge about how other things work, that I am able to just make connections to other things I have already learned; not necessarily a full blown new analogy but being able to relate certain concepts to others which I guess is an analogy in and of itself.

  I have always thought analogies to be effective in learning new material and I feel that the lack of my opinion changing over time is attributed to the fact that they work. You can't change my mind about them because I have seen the definitive proof that they work. In this class, I got a [VERY GOOD SCORE] on program4 which used semaphores and mutexes and I had my Factorio train analogy for that
- I feel like analogies are much more effective in the upper level course. Describing a thread a some relatable object performing a task is super useful in terms of understanding thread behaviors and more specifically their relations to each other.
- I definitely use analogies more frequently on a day to day basis now that I have learned the power of using them, especially when communicating with another individual who may not come from the same technological backgrounds as I. This allows me to convey big picture ideas, without getting into the minutia.
- I feel as though as time has gone on analogies have become more and more useful to help describe these complex ideas in computer science. At the lower level classes it is easy to just mentally "brute force" your way into understanding every part of a simple topic. At higher levels you have to generalize a lot more and only really memorize the overall functionality of things rather than get caught up on the really obscure cases.
- As an experienced programming/CS student, I've found analogies to be useful more for abstract, but simple, concepts like Classes more than concrete "patterns" that emerge naturally from a problem like server/client interactions. An example of a good analogy in this class would be calling a Mutex a lock or a Semaphore a timer/gateway. A bad example would be something like the rollercoaster pattern, which instead of providing a short, concise example to parallel the usage of the involved parts (Mutex-¿Lock) creates a story which is not an everyday occurrence and has a lot of intricate parts to remember (Rollercoaster, riders, rollercoaster controller, story, sequence of events).
- Yes, they help a lot more than they used to. I use them more, in most aspects of my collegiate career. I think the thing that changed my perspective is the fact that if I need to help someone or explain to someone my problem, and they don't know what is going on a lower level description is helpful. This is so they can relate what is going on to something they know, and/or understand better. Since I am a member of [ENTERPRISE GROUP] explaining

concepts such as reverse engineering to students who don't necessarily understand what is going on an analogy is useful.

- I still use analogies all the time. I think they can be really effective to help when wither trying to understand a concept, or putting that concept into practice. It is true that as I learn more, some of the analogies get ore complex, but they are still useful for keeping things straight at a high level. They start to crumble when you really try to apply them in any more detail than high level stuff, or so I've seemed to observe.

- I think analogies are more important as an upper level student. This is because early in college the syntax is more difficult than the problem. As an upper level student I am more aware of a programming languages syntax, but the problem is much more complex. This means that the analogies I use to understand the larger problem are almost more important because at this point my understanding of the programming language often means the program "writes itself" once I can grasp the problem.

- As an upper-level student, my perception and use of analogy has definition changed. I find myself using analogies a lot more frequently when trying to solve particularly difficult problems. When I am not certain of a topic, I try to find an analogy that fits the definition as well as I can. They make it easier to break whole concepts down and really understand a subject step by step. I feel they are more effective now than they were in my lower-level courses because those courses did not really have a lot of big, complicated concepts. In my upper-level courses, the subjects became more complex and it took a completely new way of thinking to really grasp. I think just have professors that try to break concepts down by using simple analogies has impacted the changes in my perception.

- The use of analogies has definitely improved by view of how to approach a problem. Instead of spinning my wheels and not making any progress by staring at my code, taking a step back and saying the requirements of the project out loud and in a context like humans competing for things on a shelf (and talking to each other to Wait() and Signal() them) forced me to slow down and try approaching the project/problem from a different angle. Sometimes it just takes a change of mindset or context in order to have a revelation about an issues and all of a sudden be able to solve it. The reason I feel that analogies are useful is because they often forced me to start over from scratch when it came to thinking about the checks, Wait()s, and Signal()s that my code needed to do. When I was simply looking at my code and trying to figure out the problem, I was only looking at what was there in my .c file at that moment. Analogies, by making me take a step back and start over, lead me to look and think instead about the things that my code did not have because when writing the code, I was thinking in terms of code (and not in terms of plain English, where sometimes the need for additional conditions may be revealed).

- I feel like I understand the importance of using analogies when trying to explain something to someone. I would say I use them much more now when explaining things to people than I did in the past. I feel this way because the first year or two of learning computer science was very difficult for me due to my inability to understand abstract concepts. I found it much easier to understand concepts abstractly after I was able to use the concepts in a concrete way.No

- As an upper-level student, I find it hard to relate concepts of computing or discrete mathematics since they do not always have REAL world examples. I think they are a very effective way of describing and relating a topic to a student, especially if you can use an example to show parallels. I guess I did not have much of an opinion of analogies until I became a [UNDERGRAD MENTORING ROLE].

- I don't feel analogies help me personally. The concepts I'm learning now are more complicated so analogies must be stretched to effectively portray a concept. I think it's easy for analogies to lead to false conclusions as they can not accurately encompass all aspects of concepts. I don't believe analogies have a place in large computing concepts.
- I feel that I have used analogies more as an upper level student. A lot of the upper level concepts seem to be more abstract or harder to understand. So I find analogies to be a valuable tool to understand more difficult concepts. This means I find them to be more effective because a lot of the upper level concepts are harder to grasp initially. And once you get the ball rolling on a concept the rest seems to come much easier. As for changes in my perception I feel like I tend to compare a lot of abstract computer concepts to real world examples. I find it easier to visualize different problems based on real world objects or examples. I learn a lot by hands on application so anything relating to real world applications tends to help me much more than text examples. That is the best explanation I can give for my shift in perception.
- I feel that when I manage to make them, they are very helpful. It is somewhat difficult to make a good analogy for something because you have to be able to cover all of the complexity of the problem you are talking about while also still being able to relate it to what you are working on. Sometimes they just happen to come to me. For example, I did [MENTORING ROLE] session leading for discrete and analogies were helpful when trying to explain things to students especially when it came to some of the labs. Overall, analogies are very helpful if they are relevant and well thought out otherwise they can be tedious and confusing.
- I feel like if anything I use analogy more as an upper-level student. This may be due to the abstract/high level topics that I try to conceptualize to better understand/work with the problems. I think the way that computer science is taught is also a factor, as my initial programs used generic variable names, but becoming more descriptive made the programs much easier later on.
- I find myself using analogies to communicate various problems to others whom either don't understand certain specifics of a problem or aren't educated enough to understand the certain technical concepts that may be used.
- I tend to be a lot more methodical early on in a programs cycle where I look at how the key mechanics of the thing I need to do work before jumping it.
- I feel they are valuable, especially for things that are very counter intuitive, such as the baton passing at first.

Appendix M

Glossary and Definitions

abstraction: Generalization of specific events and examples to a schema, allowing for more complex thought and problem solving.

analogical encoding: The process by which one derives a perceived common structure through comparison of two items or situations.

analogical reasoning: Any way of thinking in which analogy is a necessary element of the process, especially in problem solving.

analogy: A comparison drawn between two situations or items, which are deemed “analogs”. Often considered in form “X is like Y”.

class inclusion: To belong to a subset within a set — may be exemplified by the “IS-A” relationship, ie, a Doberman Pinscher IS-A Dog. The Doberman-Pinscher is a subset, and thereby an inclusion to the class of Dog.

cognitive load: Requirement of working memory resources, usually in task completion.

conceptual model: A model of a target system, such as an explanation, diagram, or abstraction, that is created by experts with the intention of facilitating understanding of the target system.

Concurrent: An upper level course offered at our university in which learners investigate topics of concurrency in computing including mutexes, semaphores, resource allocation, race conditions, and more.

context: The encompassing circumstances and environment necessary to fully understand and assess a situation.

critical pedagogy: Learners work to deeply investigate and interrogate root causes of problems within not only their domain, but the world around them.

crystallized intelligence: An aspect of the Cattell-Horn-Carroll (CHC) Theory of Intelligence.

Generally defined as prior knowledge one possesses, which may include skills, process steps, facts, and so on.

CS1: A first introductory computer science course at a degree-seeking level. Typically covers foundational ideas in computer science, and begins exploring relevant topics to future courses.

DIALECTIC: Investigating different points of view or multiple perspectives on an issues, usually by engaging in a dialogue of some form. The process intends to interpret the truth from conflicting ideas by finding compatibility through the engagement.

DISCORD: Refers to the application Discord, a VoIP application traditionally used by videogame players, but has found broader appeal. Allows for text chat, screen and file sharing, voice chat, direct message, and creation of custom rooms (servers) which can have several threads (channels).

DOMAIN: Discipline or space in which an example or idea comes from.

EPISTEMOLOGY: studies and science concerning the theory of knowledge and our justifications and beliefs.

FLUID INTELLIGENCE: an aspect of the Cattell-Horn-Carroll (CHC) Theory of Intelligence. Generally defined as the ability to solve a problem without the use of prior knowledge.

HAND TRACING: in programming, the act of executing code "by hand" (usually with pencil and paper) in order to track values and understand instruction execution line by line.

HCI: Human-computer interaction (acronym).

HERMENEUTICS: Theories of interpretation and understanding.

HERMENUETIC LOOP: A cycle describing a whole and its parts. The whole cannot be understood without inspection of each of its parts, but the parts do not make sense without the context of the whole. This results in consistent analysis and revision of understanding as vision of the parts and the whole are modified.

HUMAN-COMPUTER INTERACTION: A field of study concerning the design of computing technology, systems, and interfaces in order to understand and optimize how humans interact with computing artefacts.

INTUITION: To be able to correctly assess a situation without needing to engage critical thinking or other conscious cognitive resources.

LEVELS OF ABSTRACTION: The spectrum of representational methods and generalizations one may be working through, especially during the problem solving process. In Computer Science Education, Cutts et al [25] characterized three main levels of abstraction that one moves between at various intervals of the problem solving process: “normal language”, “CS speak” (utilization of jargon and pseudocode-esque natural language), and “executable code”.

LEXICON: Vocabulary a person, language, or discipline uses. In Computer Science, referring to both the overall discipline as well as syntactic nuances of specific programming languages.

MAPPING: An association that is created between elements of one set to elements of another set.

MEA: Means-ends analysis (acronym). An analysis process that allows one to refine their problem solving approach. An action that can be repeatedly applied to a current state to move closer to a goal state is identified.

MEME: A behavioral or cultural element that is shared with others. On the internet: digital artefacts (typically images, videos, or text) that can be modified and shared widely among groups, with individuals sharing a common understanding of meaning and intention.

MEMETICS: Studies surrounding information and culture, specifically, how information can be transmitted through cultures.

MENTAL MODEL: A person’s thoughts as to how a particular result is produced in the real world from the interaction of parts and ideas. This model includes representations of the world the person possesses, relationships they define between component parts of those relationships, and perceptions they have about actions and reactions within that system.

METAPHOR: A situation or idea is considered to be a representation of another piece of knowledge. Often invites analysis of conveyed meaning, as the representation may not be readily apparent. Typically of the form “X is-a Y”.

NOTIONAL MACHINE: An idealized abstraction of program execution and programming language semantics within computer science education, used for pedagogical purposes. See **conceptual model**.

NOVICE: In this work, a beginning programmer who is working toward a general model of foundational programming concepts.

OBSERVER CONCEPTUALIZATION: An observer’s model based on their observations of what another person’s mental model may contain. The observer’s model is distinct from the person’s mental model, but is based on observation of the person’s process, dialogue, artefacts, and so forth. It is a model that encapsulates beliefs about another’s mental model.

OPAL: Outlining Programming Analogy Layout (acronym). A developed evaluation tool resulting from this work that is meant to help instructors build out the structure of created analogies.

ONTOLOGY: Understanding the world and the nature of being or existence through categorization of entities into categories, such as the categorization of turtles to the grouping of living beings, and the subgrouping of reptiles.

PARSIMONY: The minimum effort or resources are exerted by an individual or entity — it is difficult to engage the utilization of more.

PEDAGOGY: Theories of teaching. Also, the methods an instructor applies in order to facilitate teaching.

PHENOMENOLOGY: Studies and science that specifically focus on a person's lived experiences and their effects on that person.

POLYSEMY: A word or phrase having multiple potential meanings. Often used specifically in conjunction with multiple meanings being able to be associated in the present context or domain.

PRAXIS: The process through which one applies learned information and ideas.

PRECONCEPTION: An idea one possesses prior to engagement with an activity or idea; an assumption.

REPRESENTATION: A depiction, rendering, characterization, or description of a specific thing. Not the thing itself, but illustrates the expectations of the thing.

RUBBER DUCK: In programming, a method of debugging whereby a programmer explains what their code should do and what they have written to a rubber duck (or other object, though usually one with a face for the illusion of a dialogue partner). Through the process of explaining even to a non-conversational partner, the programmer may identify the issue.

SCHEMA: The representation of a model. In cognitive science, a representation outlining thinking and behaviors, especially with regard to a certain topic.

STRUCTURE MAPPING: A theory of analogical reasoning developed by Dedre Gentner that describes analogy as a system of relations that are mapped between base and target domains.

SYMBOL: See **representation**.

SYSTEMATICITY: Holding a representation or understanding of structural relationships between entities.

TARGET SYSTEM: A phenomena or process within the world. Typically used in association with the desire to induce understanding regarding its workings.

TRANSFER: Knowledge being available for re-access in appropriate contexts by a learner.

USER EXPERIENCE: As a field of study, the exploration, understanding, and optimization of a person's actions, attitudes, emotions, and results during their engagement with a process, product, place, or other form of experience. Often an integral aspect of successful human—computer interaction.

UX: User Experience (acronym).

M.1 Chapter "Flavor" Titles Glossary

Each chapter's title began with a term that is related to the topic of glass, mosaics, or sight. This glossary identifies the meaning of these flavor titles, as well as the reason they were selected.

Transparency: (Chapter 0) The ability for light to pass through so an object can be distinctly seen. The context of the computer science discipline and this research must be presented so the research can be viewed with such context in mind.

Gazing: (Chapter 1) To look at intently in admiration or thought. The introduction outlines the aspirations of this work and highlights key ideas to be explored.

Visions: (Chapter 2) The ability to see. The way we process information and our methods of reasoning modify how we see the world around us.

Connected Shards: (Chapter 3) Fragments of glass which are fitted together to form the mosaic. Analogy is a tool that can allow us to connect existing knowledge fragments and models to new ideas.

Opus: (Chapter 4) A "creative work"; the unique way one sorts pieces to create a mosaic. OPAL is my "creative work" in approaching analogy construction. Hopefully, it can be used to inspire more analogy opuses among instructors.

Tesserae: (Chapter 5) Materials used in the creation of a mosaic. Engagement requires the use of relevant, tangible ideas for the learner, but can encourage them to use these as material to connect ideas within their mental models.

Emblema: (Chapter 6) A detailed decorative mosaic that is included in a less complex design. Memes allow the communication of complex ideas through simpler relational structures for learners to understand. Further, they can be an enjoyable "decorative" aspect of the learning environment.

Prisms: (Chapter 7) Glass or crystal pieces that refract light at specific angles, distorting its appearance. Each learner engages with the classroom and sees their experience and the material differently.

Dispersion: (Chapter 8) The refraction of colors via a prism; also: the act of distribution — incorporating a bit of polysemy. This chapter summarizes the contributions of this research and expands its focus into a spectrum of refracted research pathways.