IDENTIFYING COMPUTATIONAL PRACTICES IN INTRODUCTORY PHYSICS

By

Michael J. Obsniuk

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Physics — Doctor of Philosophy

2019

ABSTRACT

IDENTIFYING COMPUTATIONAL PRACTICES IN INTRODUCTORY PHYSICS

By

Michael J. Obsniuk

Introduction, Background, Context, Motivation, Analysis, Discussion, Conclusion.

TABLE OF CONTENTS

LIST OF TABLES

vi

LIST OF FIGURES

Chapter 1

Introduction

Since the advent of relatively inexpensive and powerful computers, researchers have been interested in their use as both professional and pedagogical tools. Their ability to quickly and precisely perform numerical calculation makes them well suited for modeling and solving modern problems in the STEM fields. Similarly, their ability to easily generate meaningful visualizations makes them well suited for the communication of scientific information. For these reasons, computation is indispensible in modern scientific pursuits.

Computation, or the use of computers to analyze complicated problems, continues to grow in many fields, from mathematics to biology. Given its utility in these professional domains, the task of effectively training students in computation has risen to the forefront of education research. This task has been shown to involve many challenges, as there are many and varied skills and pieces of knowledge that students must develop a mastery of in order to effectively utilize computation. Still, the desire to integrate computation into the STEM curriculum is stronger than ever.

While using computation to solve complex physics and engineering problems, practitioners often engage in various computational practices. Computational practices can be defined as a synthesis of computational knowledge and computational skill – highlighting the importance of being able to put theoretical ideas to practical work. Although knowledge and skill alone are important, being able to combine the two into an effective *practice* is even more

so. Although attempts have been made to define computational practices broadly, they are still lacking clear and precise definition within many particular domains (e.g., computational physics). Accordingly, this thesis focuses on identifying the common computational physics practices that students engage in while solving realistic physics and engineering problems.

There are a number of reasons for focusing on computational physics and its associated computational practices. Perhaps most important is that there is a high demand for computational skills in the workplace for physics graduates [1]. Being able to effectively prepare students requires in-depth research to develop best practices. Modern physics curricula should reflect the modern practices of professional physicists, and computation is often seen as just as important as theory and experiment. For this reason, 51 % of faculty from physics departments call for more computation in the curriculum [14].

Additionally, it is believed that students of computational physics gain a deeper understanding of the physical concepts [12, 29]. Visual packages such as VPython or Glowscript [40] allow novice programmers to create stunning three-dimensional visualizations that allow them to more easily interact with the fundamental concepts.

Further, computation allows for the analysis of realistic problems that have no closed-form solution. Its ability to numerically integrate supports a more exploratory approach to analyzing physical systems and learning physics. That is, the repeated application of Newton's second law allows for a more general analysis. This more exploratory approach is thought to encourage students to construct more realistic and accurate computational models through computational thinking [1].

Finally, computational thinking is a term that has become increasingly popular since its introduction in the early 1980s. This term, although frequently used today, is difficult to concisely explain given its many and varied definitions. Even within the fields of education

and computer science, many different viewpoints exist on the topic, and the corresponding definitions are just as varied [22]. However, many of these definitions share one fundamental characteristic: solving complex problems through abstraction and analytic thinking with the aid of computer algorithms, which is precisely the type of thinking that this thesis explores.

Computational thinking is so highly valued by the modern enterprise of science education that the Next Generation Science Standards (NGSS) laid out a framework for identifying computational thinking in K-12 settings. As early as the fifth grade students are expected to be able to think computationally. They describe computational thinking, at this level, in terms of analyzing data and comparing approaches. By the time students reach middle school, computational thinking advances to analyzing large data sets and generating explanations. Finally, in high school, computational thinking expands to constructing computational models and using them to answer questions [32]. Clearly, computational thinking is a complicated concept which requires substantial explanation.

Experts in the field still have a ways to go when it comes to clearly defining computational thinking within science education, and, within physics education, specifically. However defined, though, this type of abstract and algorithmic thinking is pervasive – it extends beyond computer science into fields from geology to astronomy, and even beyond STEM [9]. It is becoming increasingly clear that "computational thinking is a fundamental skill for everyone, not just computer scientists [48]."

Given the recent interest in scientific practices, and computational thinking more specifically, a taxonomy of the computational practices indicative of computational thinking has been proposed [45]. This taxonomy, comprised of twenty-two individual yet inter-related practices, fitting into four different categories, is meant to help guide instructors and researchers as they attempt to teach and better understand computational thinking in science

classrooms. Each practice, according to the taxonomy, is defined broadly and from an expert level so as to be applicable to a wide range of science classrooms.

However, the broad and expert-generated definitions that make the taxonomy widely applicable also leave it relatively vague and difficult to apply to any particular situation. Reducing the vagueness and difficulty of applying this taxonomy to a specific domain of inquiry (i.e., introductory physics) is a challenging but important task. Having a taxonomy that is both precise and easy to apply will provide a solid foundation for instructors to generate/validate computational problems and for researchers to analyze the learning process. **Accordingly, it is important that we identify, through direct observation, the set of computational practices that are common to computational introductory physics.** This involves not only identifying the practices, but also the underlying knowledge and skills.

Ultimately, this dissertation is meant to illustrate the process of identifying the common practices that groups of students engage in while solving a realistic computational physics problem. In Ch. 2 we explicate the prior research on computation and its results, as well as the theoretical and methodological underpinnings of the study. This includes the historical and recent results from Physics Education Research (PER) and Computer Science Education Research (CSER). In Ch. 3, we describe the course from which our data has been collected – a calculus-based introductory physics course with a focus on engineering, working in groups, and computation. We also describe the types of computational problems students are working on in class. In Ch. 4, we provide a motivation for not only the existence of the study, but also the theories and methods that we decided on using. Our theories and methodologies used depended highly on the type of data that we had and the type of research we were conducting. In Chs. 5–7, we present the analysis and results of our current study

4

with concluding remarks.

Chapter 2

Background

In order to better understand the analysis and results of this thesis, there are three broad and underlying topics that deserve elaboration. First, the concept of computational thinking and its definition. Next, the results from Physics Education Research (PER), including the various implementations of computational physics and its effect on learning. Finally, the qualitative methodologies and the framework that we have used to guide our analysis.

2.1 Computational thinking

As mentioned in the introduction, computational thinking and its associated practices within introductory physics are of primary interest to this thesis. These practices, which are generally thought of as a combination of the accumulation of knowledge and its application through particular skills, are the observables that we can look for within our data. Building on previous research that attempts to tie computational thinking to observable skills and practices [1, 32, 45], we have attempted to more clearly and precisely define the fundamental practices within introductory physics.

The history of computational thinking and its defintion is long but incomplete [34, 35, 48, 47, 3, 22, 9]. Early on, the term was introduced by Seymore Papert as it related to students actively constructing knowledge through the production of an artifact (ideally, but not necessarily, a computer program). This idea of learning through construction, often

called "constructionism," was built on the Piagetian idea of "constructivism." Constructivism states that students learn best when they are actively involved in the construction of their knowledge [37]. Constructionism, on the other hand, believes that it is the construction of a tangible (or intangible) object that is of critical importance when actively constructing knowledge [34].

Papert was very interested in looking at how computers could be used to teach things to students. Some of his earliest research into Logo (an educational programming language aptly named for its focus on reasoning) and its use as a learning tool focused very heavily on the construction of two-dimensional shapes on a computer screen [33]. The computational power to be able to generate these accurate and accessible visualizations while forging new ideas made Logo and other similar computational implementations very powerful for learning.

However, Papert did not initially attempt to define computational thinking in terms of constructionism. Rather, he commented that attempts to integrate computational thinking into everyday life had failed because of the insufficient definition of computational thinking. He optimistically claimed that more attempts to define computational thinking would be made, and eventually "the pieces will come together [34]." Papert would later go on to say that computational thinking involves "forging new ideas" that are both "accessible and powerful [35]."

More recently, building on Papert's preliminary observations, Jeanette Wing defines computational thinking as it relates to the processing power of modern computers with the addition of human creativity. This echoes the core sentiments expressed by Papert of using human creativity to forge new ideas that are computationally powerful. She states that "[it] involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. [48]"

Wing is careful to remind readers that computational thinking is a fundamental skill for everyone, not just computer scientists [47]. This speaks to the robust nature of computational thinking, but also speaks to the difficulty in clearly defining it. She believes that computational thinking should be taught at the introductory college level, and should even go so far back as to be introduced at the pre-college level. Wing makes substantial progress in defining computational thinking, but still falls short – especially within particular subdomains like computational physics or chemistry.

Further elaboration by Alfred Aho points out that the process of finding the right tool (e.g., a software like Excel or an algorithmic model like Euler-Cromer) for the right job is a clear indicator of computational thinking. He considers computational thinking to be the " thought processes involved in formulating problems so their solutions can be represented as computational steps and algorithms." Mathematical abstraction (sometimes called modeling) is at the heart of computational thinking, and being able to choose between competing abstractions (models) is of critical importance [3]. Aho points out that although there are many useful definitions of computational thinking within the field of computer science, new domains of investigation (e.g., introductory physics) require definitions of their own.

Aho believed that clear and precise definitions of computational thinking within a particular field were required for practitioners to be able to leverage them in their classrooms. Ideally, these definitions would match the various models used within that particular field. For example, within cloud computing, the various models used while developing systems and building tools could be extended to research [3].

Theoretical definitions aside, The Next Generation Science Standards has most recently attempted to operationalize a definition of computational thinking in K-12 science classrooms. They have included computational thinking as one of their core practices, and

identify a handful of expectations for K-12 students that require computational thinking. According to the NGSS, students should be able to [32]:

E1. Recognize dimensional quantities and use appropriate units in scientific application of mathematical formulas and graphs.

E2. Express relationships and quantities in appropriate mathematical or algorithmic forms for scientific modeling and investigations.

E3. Recognize that computer simulations are built on mathematical models that incorporate underlying assumptions about the phenomena or system being studied.

E4. Use simple test cases of mathematical exprsesions, computer programs, or simulations–that is, compare their outcomes with what is known about the real world – to see if they "make sense."

E5. Use grade-level-appropriate understanding of mathematics and staistics in analyzing data.

These expectations, though useful, are rather broad and can be reasonably applied to any science classroom. For example, the expectation of being able to recognize dimensions in a mathematical formula (E1) might show up in a chemistry classroom focusing on mass conservation before and after a chemical reaction. Alternatively, the expectation of students understanding that simulations rely on mathematical models (E3) might show up in a biology course involving predator/prey predictions based on an underlying computational algorithm (e.g., the Lotka-Volterra equations).

Although these expectations require computational thinking, they are still rather vague and could apply to any number of different science classrooms. More clearly and precisely

defining these expectations is an important task, especially within a particular domain of interest. Without precise and domain-specific definitions, applying them to a particular classroom is rather difficult for practitioners. Accordingly, one field whose precise definitions are particularly lacking (though, progress is being made on) is that of physics.

Introductory physics is a field whose problems are ideal for a computational analysis. The various models that are used in introductory physics (e.g., A newtonian gravitational force model, an Euler-Cromer Newtonian integration algorithm, non-linear drag model) can be used to predict the motion of realistic and complex mechanical situations. This type of realistic problem solving is a desirable skill to train students in, and it represents a practice that is authentic to the field. Accordingly, it is of critical importance that we work to clearly and precise this and other practices within introductory physics.

Similarly, although defining computational thinking within K-12 is an ideal starting point, it should also be extended to more advanced levels. There are many concepts requiring computational thinking that are unique to the university level and above, and as students advance throughout their educational career, it is important that we study them. To wit, the AAPT Recommendations for Computational Physics in the Undergraduate Physics Curriculum has identified the skills (physics-related and technical) and tools that should be included in a modern physics curriculum [1]. These recommendations include roughly ten skills like debugging, testing, and validating code and many tools like Excel or Python.

Still, More research is needed to not only more clearly define the computational practices observed in introductory physics, but also to more clearly understand the habits of mind and types of thinking that students are engaging in. It is important that we further define expectations around computational thinking within a particular domain of interest (i.e., introductory physics) and at a particular level (i.e., university calculus-based).

2.2   Physics Education Research

This section focuses on the development of the different implementations of computational physics problems (e.g., BOXER or Glowscript) [17, 39, 14, 31], the results from DBER (e.g., student challenges) [12, 8, 6, 24, 15, 44], and most importantly the remaining questions.

2.2.1   Implementation

The focus on computational thinking in Physics Education Research (PER) has been increasing over the past decade. Historically, computation as a pedagogical tool has taken many forms, but its implementation has usually focused on two things: its ability to handle tedious calculations and its ability to generate precise visualizations.

For example, one of the earliest forms of computation at the introductory level, called BOXER, used "simple programming" to generate two-dimensional shapes on a computer screen. This "reconstructible medium" allowed even novice programmers to take advantage of the processing and visualization power of computers. To illustrate, Fig. 2.1 shows the graphical user interface for a program in BOXER that is meant to generate a star and a triangle for two different objects. The underlying algorithms are laid out in sequential steps.

Alternatively, another implementation of computational physics takes the name VPython: the Python programming language with the Visual module. Historically, the ultimate goal of developing VPython was to "make it feasible for novice programmers in a physics course to do computer modeling with 3-dimensional visualizations [40]."

Although VPython was ideal for novice programmers, it also catered to more advanced users. Its basic algorithm is an Euler-Cromer style integration to calculate the contstantly updating position and momentum (or velocity) of an object within a while loop that depends
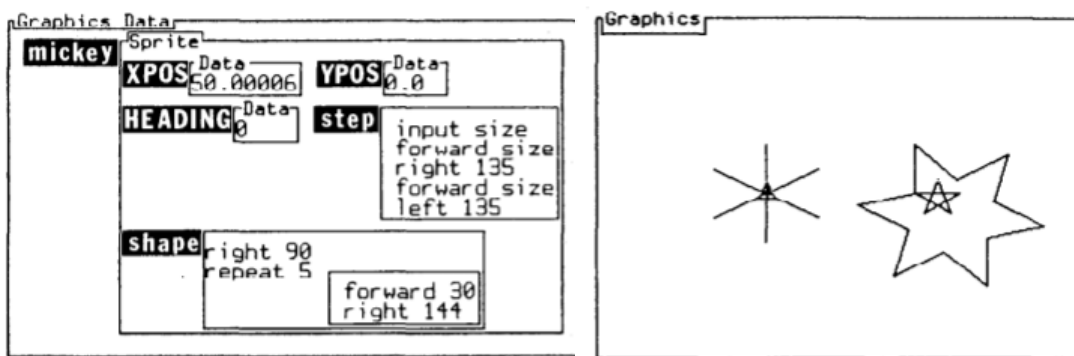
Figure 2.1: Graphical user interface for BOXER showing the graphics data (e.g., x-position, y-position, step instructions) and the resulting graphic for a sprite named mickey.

on time. For example, Fig. 2.2 shows the basic structure of a very simple but powerful MWP. This Euler-Cromer algorithm can be used to analyze very simple situations (e.g., free-fall motion) as well as more complicated and realistic (e.g., the motion of satellites and rockets).

Along with the development of VPython, a software called Easy Java Simulations (EJS) was increasing in use [18]. These simulations were meant to give students a little more control behind the scenes, like VPython, while still limiting the generalizability like PhET simulations (described below).

For example, a simulation of a pendulum could be constructed in EJS by dragging a particular object (e.g, a pendulum bob) into the model and using their built-in editor to solve the associated differential equation (see Fig. 2.3). Only a small amount of modification is needed, reducing the load on novice programmers. This reduction in load through scaffolded programs is very similar to the MWPs used in a lot of the research within PER [43].

Another implementation of computation, frequently used today, are the Physics Education Technology (PhET) simulations [36]. These simulations have realistic graphics that display buttons, sliders, and knobs that can be graphically tweaked to change parameters in a system. This type of testing – searching for the effect on a physical system with the

```
1   bead = sphere(pos=vector(0,0,0), radius=0.1, color=color.red)
2
3   bead.m = 0
4   bead.q = 0
5   bead.v = vector(0,1,0)
6
7   g = vector(0,9.81,0)
8   E = vector(0,0,0)
9
10  Fg = -bead.m*g
11  FE = vector(0,0,0)
12
13  Fnet = Fg + FE
14
15  bead.a = Fnet/bead.m
16
17  t = 0
18  tf = 10
19  dt = 0.01
20
21 ▾ while t < tf:
22       rate(100)
23
24       bead.pos = bead.pos + bead.v*dt
25       bead.v = bead.v + bead.a*dt
26
27       t = t + dt
```

Figure 2.2: A MWP illustrating that the basic control structure (while loop) and integration algorithm are pre-written so that students can focus on the computational force model that must be constructed in line 11.
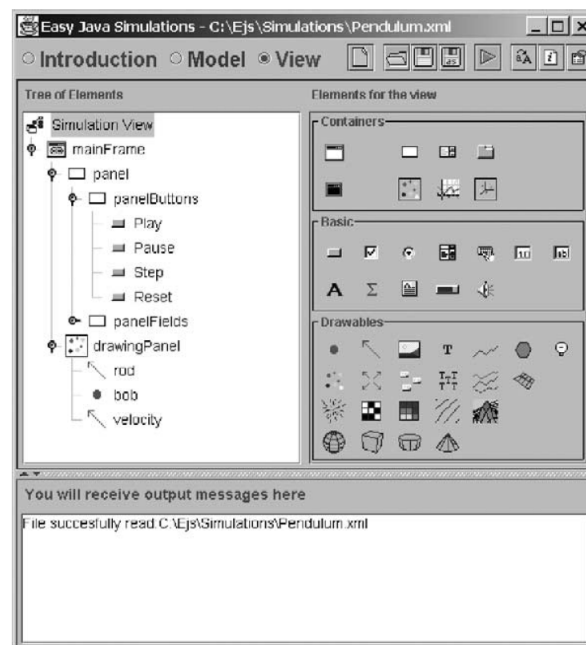
Figure 2.3: Graphical user interface for an EJS illustrating the "drag-and-drop" nature of the software. Elements (e.g., a pendulum bob) can be added or remove from the different panels (e.g., the drawing panel) in the simulation view.

Figure 2.4: A PhET simulation illustrating the dependence of pendulum motion on the length of the pendulum, the mass of the pendulum bob, the magnitude of the local acceleration due to the gravity, and any frictional forces.

variation in a parameter – is meant to be more engaging and conducive to learning.

For example, the PhET simulation shown in Fig. 2.4 is meant to demonstrate the dependence of a pendulum's motion (e.g., its period or amplitude of oscillation) on the various parameters of the system (e.g., the length of the pendulum or the magnitude of friction). Being able to hold one parameter constant while varying the other helps students to confidently identify its qualitative effect.

Finally, one of the more recent implementations of computation at the introductory level is called Glowscript [12]. Glowscript is a variant of VPython which is designed, in part, to easily generate three-dimensional visualizations. For example, the rather complicated Glowscript program shown in Fig. 2.5 uses an inverse-square electric field model with for and if loops to generate a visual representation of the electric vector field at any point in space surrounding a discrete charge distribution.

This more realistic and descriptive three-dimensional visualization leveraged by Glowscript

```
1  GlowScript 2.7 VPython
2
3  scale = 4e-14/1e17
4  ec = 1.6e-19   # electron charge
5  kel = 9e9       # Coulomb constant
6  scene.range = 2e-13
7
8  charges = [ sphere( pos=vector(-1e-13,0,0), Q=ec,  color=color.red,  size=1.2e-14*
9              sphere( pos=vector( 1e-13,0,0), Q=-ec, color=color.blue, size=1.2e-14*
10
11 s = "Click or drag to plot an electric field vector produced by the two charges.\n
12 s += "On a touch screen, tap, or press and hold, then drag.\n"
13 s += "Arrows representing the field are bluer if low magnitude, redder if high."
14 scene.caption = s
15
16 def getfield(p):
17     f = vec(0,0,0)
18     for c in charges:
19         f = f + (p-c.pos) * kel * c.Q / mag(p-c.pos)**3
20     return f
21
22 def mouse_to_field(a):
23     p = scene.mouse.pos
24     f = getfield(p)
25     m = mag(f)
26     red = max( 1-1e17/m, 0 )
27     blue = min(  1e17/m, 1 )
28     if red >= blue:
29         blue = blue/red
30         red = 1.0
31     else:
32         red = red/blue
```

Click or drag to plot an electric field vector produced by the two charges.
On a touch screen, tap, or press and hold, then drag.
Arrows representing the field are bluer if low magnitude, redder if high.

Figure 2.5: Glowscript output demonstrating its ability to generate three-dimensional visualizations of objects, vectors, and graphs. The ability to quickly and accurately generate three-dimensional vectors allows for more flexibility and a deeper understanding of, for example, electric (vector) fields.

and VPython is thought to encourage students to form a deeper understanding of the underlying physics concepts. Although many different implementations of computation exist [33, 17, 36, 12], research focusing on improving those implementations in PER is still lacking. Some of the critical results, though, are described below.

2.2.2   Results

In the early 2000s, Chabay began to research the integration of computation into the introductory calculus-based physics course using VPython [12]. This course included a computational curriculum following that presented by *Matter and Interactions*. Primarily, the courses studied by Chabay focused on the application of the integral equation governing the linear motion of objects (i.e., $d\vec{p} = \vec{F}_{\mathrm{net}}\, dt$ and $d\vec{r} = \vec{p}/m\, dt$). These equations were applied iteratively through an Euler-Cromer style integration algorithm, and allowed a more thoroughly analysis of position-dependent forces (e.g., the Newtonian gravitational and spring

forces).

Chabay found that one of the positive aspects of including computation at the introductory level was to stimulate creativity in students [12]. This creativity in approaching problem solving is thought to lead students to the construction of more realistic computational models. In other words, computation allows students to easily verify and/or modify a model, encouraging creativity and a "guess and check" approach to problem solving.

She also found that requiring students to program at the introductory physics level was a difficult barrier to overcome. Given that there is so much content to be covered in so little time in most introductory physics courses, finding the room/time to discuss the basics of programming is difficult. One of the ways in which this difficulty is overcome is by providing Minimally Working Programs (MWPs) to students. The MWP for a particular problem usually runs without error from the start (pre-written code), and requires small (or at least localized) changes to the underlying computational models. For example, see the MWP in Fig. 2.7.

Around that same time, Kohlmyer dug deeper into student performance [29]. He found that, among other things, computational modeling students struggled to recognize that computers could even be used to solve physics problems. Furthermore, once they did decide to use a computer, they struggled with the concepts and components of creating a computational model. These results were generated from two experiments: looking at how students approach novel problems with computation and looking at the differences in the fundamental principles used as compared to traditional (i.e., non-computation focused curriculum) students.

Interestingly, he found that students decided to take advantage of the Euler-Cromer style integration in discrete form even when they weren't using a computational model. That is,

students made use of the key conceptual tool that they were taught – even if just on paper.

He also found that the complex procedure needed to model attractive position-dependent forces was a difficult challenge for students. Reducing this and other difficulties can be achieved through increasing the frequency of computation throughout the course and requiring computational homework problems. Kohlmyer made explicit the wide variety of unanswered questions that could be pursued in further research, hinting that the process of "making assumptions" and incorporating them into a computational model would be of particular interest.

In 2011, Weatherford began to look at integrating computation into the physics lab curriculum and the sense-making that students engage in [43]. His study was an in-depth qualitative analysis of group problem solving, focusing on three different contexts: a scattering problem, a spring-mass problem, and a spacecraft-Earth problem. A coding scheme was developed to help categorize different portions of transcript, as shown in Fig. 2.6.

He found, among other things, that computational physics students were able to reasonably interpret physical quantities according to their variable name. For example, the mass of a satellite might be defined as `m.satellite = 1`, or the net force acting on object may be defined as `Fnet = vector(0,-m*g,0)`. These pre-written variables are named so as to suggest to the students what physical quantity they represent. However, the more complicated the definitions get (e.g., a function of multiple variables like `Fnet = -k * (ball.pos - origin.pos) / mag(L)`, the more students struggled at recognizing it.

Additionally, Weatherford was able to encourage students to begin to incorporate a computational model in a MWP by providing a minimum level of support. That is, only omitting the fundamental physics calculations that students are meant to engage with (e.g., various computational force models) helps to keep students focused on the physics. Other tasks that
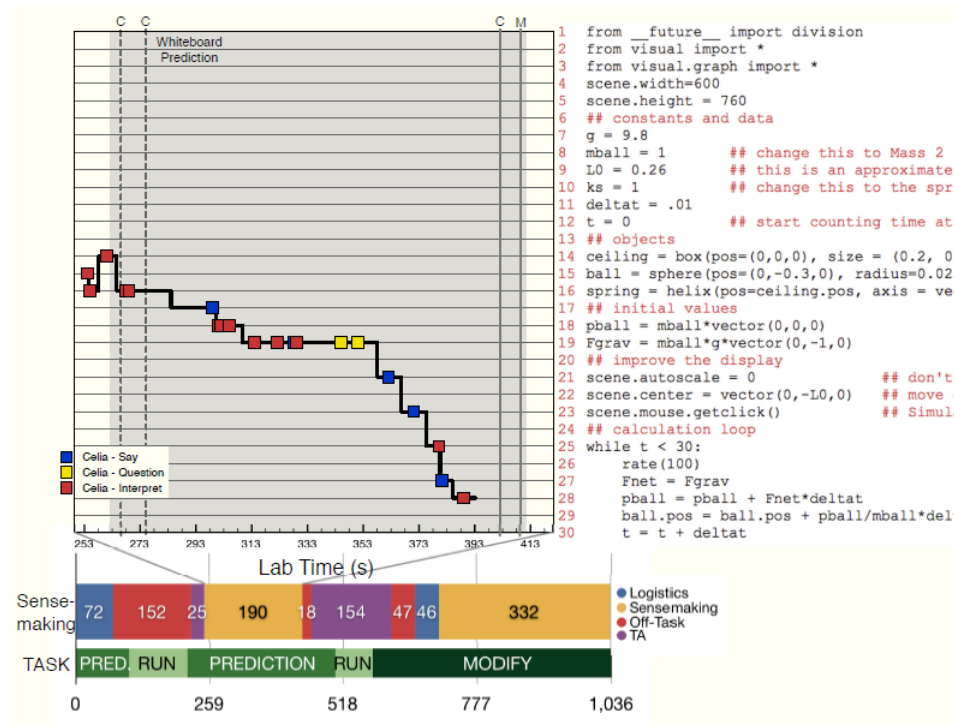
Figure 2.6: A sample of the in-depth analysis Weatherford performed. Each particular line of code that the group is focusing on is tracked in time and coded according to a scheme.

are not physical in nature have a tendency to derail the physics discussion and the problem solving process in general. For example, ensuring that the end of a spring is connected to the end of a mass in a computational spring-mass analysis begins to overshadow the more fundamental task of incorporating/constructing a position-dependent Hookian spring force. Alternatively, figuring out how to use the `mag()` function in Python can sidetrack the ultimate goal of constructing a position dependent gravitational force.

Weatherford clearly pointed out that the MWP activities in their study had much room for improvement, and that more research was needed on fostering student proficiency in computational physics. The sequence of MWPs in his study didn't quite raise students' program comprehension and program interpretation skills to a certain proficiency, but he believes that more research will shed light on the subject.

18

```
19  while t < tf:
20
21      r = craft.pos-Earth.pos
22      rhat = r/mag(r)                          Force Calculation
23      Fgrav = -G*mEarth*mcraft/mag(r)**2*rhat
24
25      pcraft = pcraft+Fgrav*deltat             Newton's Second Law
26      craft.pos = craft.pos + pcraft/mcraft*deltat    Position Update
27
28      trail.append(pos = craft.pos)
29      t = t + deltat
30
31  print 'Craft final position: ', craft.pos, 'meters.'
```

Figure 2.7: An expected solution to a computational satellite-Earth problem where the New-tonian gravitational force has been constructed from a separation vector and its magnitude. The force calculation has been incorporated into the momentum through Newton's second law, and the momentum is incorporated into the position through a position update.

In 2011, Caballero was able to identify a number of frequent student mistakes which were grouped into three different categories: initial condition mistakes, force calculation mistakes, and second law mistakes [10]. An initial condition mistake might take the form of an incorrect initial velocity or momentum of the satellite. A force calculation mistake might manifest in a constant spring force rather than a position dependent spring force. A second law mistake might involve missing the division of the mass from the net force on an object so that the velocity is correctly updated according to the acceleration. These frequent mistakes result in both unexpected and physically inaccurate visualizations.

Based on his analysis of the satellite-Earth problem, shown in Fig. 2.7, he concluded that the majority of students ($\sim 60\%$) were able to correctly computational model novel physics problems and that the practice of debugging would serve students well. Particularly, the act of troubleshooting syntax errors as well as the act of troubleshooting of *physics* errors.

2.2.3 Remaining questions

Although many aspects of computation and computational thinking at the introductory level have been studied, there are still many unanswered questions within physics education. Particularly, as to the types of practices students are engaging in that are indicative of computational thinking. More research is needed to not only more clearly define the computational practices observed in introductory physics, but also to more clearly understand the habits of mind and types of thinking that students are engaging in. **This thesis attempts to provide clear and precise definitions with examples of the various practices that students engage in within introductory physics.**

2.3 Framework

Recently, a framework for identifying the computational practices that are indicative of computational thinking has been proposed by Weintrop et. al. This framework was developed using existing literature on computational thinking, interview with mathematicians and scientists, and most importantly, computational activities from general science and mathematics classrooms.

In order to develop their framework, a literature review was performed to generate an initial set of 10 math and science practices. These initial practices are repeatedly cited as being central to computational thinking. For example, the broad and repeatedly cited practice of generating algorithmic solutions might require a student to engage with a differential equation algorithm. These broad initial practices were used to guide the subsequent qualitative analysis.

Using the initial practices resulting from the literature review, two reviewers indepen-

dently coded for the various "facets" of computational thinking that were required by the curricular materials. They analyzed 32 different computational activities from chemistry to programming, resulting in 208 facets which were grouped into 45 different practices.

Next, a review process incorporating feedback from multiple sources (e.g., teachers, content experts, and curriculum designers) was used to reduce the 45 practices into 27, which were further organized into 5 different categories. Further, external interviews were conducted with 16 K-12 science and mathematics teachers, helping to reduce the 27 practices into 22 fitting 4 different categories, shown in Tab. 2.1.

Data	Modeling	Solving	Systems
Creating	Conepts	Preparing	Investigating
Collecting	Testing	Programming	Understanding
Manipulating	Assessing	Choosing	Thinking
		Creating	Communicating
		Debugging	Defining

Table 2.1: The framework developed by Weintrop et. al to describe the computational practices observed in science and mathematics classrooms. Each category contains between five and seven individual practices, and each practice has between two and seven fundamental characteristics.

Finally, 15 interviews with STEM professionals were conducted to rate their framework according to its applicability to authentic professional practices and to give direction for future improvement. For example, interviews showed that the practice of testing and debugging was a crucial practice that was not adequately captured by the framework – an improvement that should be made on future iterations of the framework.

The four different categories of practices are labeled as data, modeling and simulation, computational problem solving, and systems thinking practices. The data practices focus mostly on the creation and visualization of data. The modeling and simulation practices focus mostly on the design, construction, and assessment of a computational model. The

problem solving practices focus mostly on programming and debugging, while the systems thinking practices are a little more abstract and focus mostly on the structure of the program itself.

As a more concrete example, the computational practice of creating data (a data practice) has three fundamental characteristics: the creation of a set of data, an articulation of the underlying algorithm, and a use of the data to advance their understanding of a concept. The more of the characteristics that we observe in a particular excerpt, the more confident we are that that excerpt can be classified as that practice.

Although each practice is defined like this, according to Weintrop et. al, the characteristics themselves are rather vague (similar to the operational definitions from the NGSS). For example, the computational practice of assessing computational models requires the identification of a phenomenon, a computational model, and a comparison made between the two. Although it is clear what a comparison would look like in any situation, the phenomena studied and the models used will depend greatly on the context (See Ch. 3). For this reason, much more work must be done to clearly define computational thinking within introductory physics classrooms – a central task to this thesis.

Ultimately, Weintrop found three main benefits to including computation: it builds on the reciprocal relationship between computational thinking and STEM domains, it engages learners as well as instructors, and it introduces an authentic and modern element of doing science. However, he is clear to indicate that more research is needed to better address the challenge of educating a technologically and scientifically savvy population. This thesis attempts to improve that education process by providing clear and precise definitions with examples of the computational practices that are indicative of computational thinking.

2.4 Task analysis

A task analysis is a procedure that can be used to better understand the requirements of a particular task and the way an "operator" (or group of operators) might work to satisfy those requirements [28]. This type of task analysis is usually focused on the observable actions that an operator might engage in while working toward a particular goal (e.g.,), but there is also a strong cognitive link between the observed actions and the requirements of the task [16].

Before beginning a task analysis, data must first be collected. Often, the method for collecting data is observation based (e.g., observing the actions of a group of operators as they carry out a task), although data can also be subject based (e.g., asking an expert what the ideal actions would be to carry out a task). Either way, the task itself generally guides the collection of data.

Once the data has been collected, there are many different types of descriptions that can be attached to it and just as many techniques that can be used to generate them. For example, one of the techniques frequently used is to *chart and network* the data. These descriptions can be written, but are most often presented visually through information flow charts or Murphy diagrams. This thesis leverages a technique for generating an *organized hierarchy* of description of the data: complex tasks are broken down into multiple smaller but more manageable tasks.

In order to give this research a solid foundation, early on, we conducted a task analysis of a complicated computational physics problem. We wanted to look at what students were doing in a particular introductory physics classroom, and the task analysis was necessary to help guide our qualitative analysis.

Within an any particular classroom, there are a myriad of expected and unexpected tasks that students engage in while solving a particular problem. For example, taking the time to name a variable with meaning, working to construct a multiple-variable function, or changing the color of an object within a program. Given the almost limitless number of tasks that might draw students' (and our) attention, the task analysis was used to reduce the initial set of tasks that we focused our attention on. This initial set of tasks was modified and expanded during subsequent qualitative analysis (see Sec. 2.5).

A task analysis consists of breaking a problem down into multiple smaller but manageable sub-tasks that can be tied together at the end. This type of analysis is frequently used in the fields of mathematics and computer science [11, 13, 20, 2]. The smaller but manageable sub-tasks are the "unit of analysis" that can then be searched for within data. For example, an expert group might proceed in predicting the motion of an object by first constructing an Euler-Cromer style algorithm, constructing the various forces, and then construct the initial parameters of the system. These steps can be done in any order, but are all necessary to the overarching task.

This type of process was used by Catrambone to show that breaking a problem down into smaller but manageable sub-tasks helps students to transfer knowledge to new and novel problems [11]. He and others believe that it is a heriarchical structure of tasks rather than a linear structure of tasks that students need to transfer knowledge to new and novel situations. The flexibility of a heirarchical structure is thought to support a more varied approach to solving a problem.

They performed three experiments, each focusing on how students transfer knowledge to new and novel problems. The first experiment was a comparison between the meaningfulness of a label's name. They found that the more meaningful the label was, the better prepared

24

students were to solve new and novel problems. The second was a deeper study of the connections between labels and sub-tasks. They found, to a reasonable degree, that there was a fundamental connection between labels, sub-tasks, and how they were grouped. The third was a talk-aloud study that looked at self-explanation while solving problems. They found that aptly named labels could be used to cue students to group sub-tasks and explain their purpose through self-explanation.

The task analysis of the problem that this thesis focuses on was initially constructed by a single content expert. After the first iteration it was presented to additional experts. Through the discussions surrounding these iterations, it became clear that the construction of the position dependent Newtonian gravitational force in code is a multi-step procedure involving a number of different sub-tasks. The task analysis was iteratively refined through this process until all experts agreed that the sub-tasks shown in Tab. 2.2 were sufficiently described/defined to be useful in video analysis.

On top of this expert generated solution, there are many other (both expected and unexpected) student generated solutions that we observe in the data. However, the expert generated solution is an ideal path to follow and so the instructors try to keep groups moving

Step (Sub-Task)	Associated Code
Construct separation vector between interacting objects	$\mathrm{sep\ =\ obj2.pos}$ $\mathrm{-\ obj1.pos}$
Construct the unit vector	$\mathrm{usep\ =\ sep/mag(sep)}$
Construct the net force vector	$\mathrm{Fnet\ =\ -G*m1*m2*usep}$ $\mathrm{/mag(sep)**2}$
Integrate the net force over time into momentum	$\mathrm{obj.p\ =\ obj.p\ +\ Fnet*dt}$

Table 2.2: Some of the necessary steps that must be taken when constructing a Newtonian gravitational force in code. Each step is associated with the construction/modification of a line of code.

in this direction. For example, a sufficient force model be constructed in terms of the polar and azimuthal angle of the satellite, although it requires a substantial amount of work to code. Both the expert and student generated solutions are a good place to look for evidence of computational thinking and its accompanying practices.

2.5 Thematic analysis

Thematic analaysis is a poorly defined, but commonly used, type of qualitative analysis that is predominately used within psychology. However, Braun makes the well-supported case that thematic analysis can effectively be used in many other fields (e.g., nursing or physics education) and clearly defines the sufficient steps that can be taken in order to complete a reasonably reliable and valid thematic analysis [7, 19, 5, 41, 27, 38, 4].

Within PER, thematic analysis is usually used for analyzing interview or work-aloud data of students solving problems. For example, Irving found that there were many different themes that came from the various perceptions students have about what it means to "be a physicist" [25]. These themes were then broken down into 12 sub-categories (e.g., high or low interest in research), highlighting the different perceptions students had about what it means to "do physics." This type of analysis, as demonstrated by Irving, can be used to generate robust themes that can be used to inform instructional changes/improve instruction.

However, thematic analysis is just one of many qualitative techniques that can be used to analyze qualitative data. The various qualitative methodologies can be broken into roughly two main types: those strongly tied to a theory/epistemology and those that are developed independent of a guiding theory/epistemology. Thematic analysis, according to Braun, is of the second type. So as to guard against the often cited critique of thematic analysis as being

ill-defined [4], Braun presents a 6-phase guide to conducting a reliable and valid thematic analysis.

According to Braun, a thematic analysis is a "method for identifying, analyzing, and reporting patterns [themes] within data." This method consists of 6 different phases, usually followed linearly, to finally produce a report (e.g., a thematic map) of the various themes and their relationships within a set of qualitative data. However, before entering the first of the 6 phases, there are a few fundamental decisions that must be made and explicitly stated. Ideally, these decisions will be made in relation to the research question and the goal of the study.

First, it is crucial that researchers explicitly state the metric by which they plan to identify themes. For example, a theme that shows up more frequently is not necessarily more important. Additionally, a theme that shows up less frequently is not necessarily less important. Rather, it is important to be consistent throughout anlysis. This thesis mostly focuses on the more frequent themes, but consideration is also given to themes that are particularly illustrative yet infrequent.

Second, researchers must decide between a rich description of the entire data set or a more detailed account of a particular sub-set. For example, within physics education, you might be interested in a rough description of the entire process that a group followed to successfuly solve a complicated problem. Alternatively, you might want to focus in on a particular sub-task and its nuance. Again, it is important to be consistent throughout your analysis. This thesis focuses on a more detailed account of a particular sub-set of the themes (i.e., those involving computational thinking).

Third, researchers must decide between an inductive and a more theoretical approach to the generation of themes within their data. An inductive approach often leads to themes that

are not related to the original research questions, but rather have generated spontaneously and are more strongly ties to the data itself. A theoretical approach, on the other hand, often leads to a set of themes that are less descriptive but are better suited to answer a particular research question. Again, it is important to be consistent throughout your analysis. This thesis follows a more theoretical approach, using the theoretical framework presented in Sec. 2.1 as a foundation for the generation of our themes.

Fourth, researchers must decide whether they will be looking for semantic or latent themes within their data. Semantic themes are those that are clearly indicated within the data, whereas latent themes often go beyond what is actually being observed. For example, within physics education, a group of students might be struggling with a particular problem. The reason for this struggle might otherwise go unnotticed without looking beyond the immediate and recognizing that each student had a late and mentally taxing chemistry exam the previous night. Usually a thematic analysis focuses on one level, and as always it is important to be consistent through your analysis. This thesis primarily focuses on the semantic themes that are directly tied to the actions observed during the problem solving process.

Fifth, researchers must be choose between an essentialist and a constructionist thematic analysis. An essentialist thematic analysis allows researchers to theorize student understanding and meaning in a straightforward way [38, 46]. A constructionist approach focuses more on the overarching sociocultural and structural environment that each student lives within. It is important to be consistent throughout your analysis, and this thesis focuses on a more essentialist approach, paying special attention to the computational thinking and habits of mind that students are engaging in.

Once these decisions have been made, the qualitative analysis can proceed through the
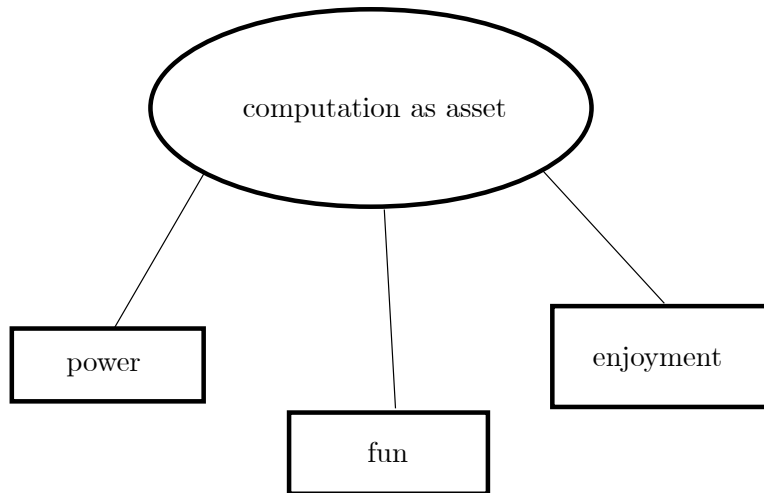
Figure 2.8: A final thematic map showing the components of a theme named "computation as asset." The main components of this theme are "power," "fun", and "enjoyable."

6 phases laid out by Braun. The first phase focuses on (1) transcribing and farmiliarizing yourself with the data. Reading through the transcripts multiple times helps to generate preliminary ideas that can be (2) coded for further investigation. Next, each code must be (3) collated with the corresponding transcript so as to provide a context. After the codes have been collated with the corresponding transcript, (4) themes begin to emerge in the third phase. Reviewing any themes that emerge, particularly against the coded extracts and the transcript as a whole, leads to the next phase of (5) defining, validating, and naming any themes. These themes can finally be presented in a (6) scholarly report with step-by-step transcript analysis and/or a thematic map. A thematic map, like the one shown in Fig. 2.8, shows not only the components of a theme, but also the *relationships* between those components.

Braun is clear to point out that there are many pitfalls associate with thematic analysis, and that researchers must be cognizant of them through every phase of the process. For example, one of the pitfalls she highlights is a possible mismatch between the data and the analytic claims that are being made. In other words, it is important to always closely tie

your claims with the actual data. This closeness of the claims to the data can be ensured through frequent inter-rater reliability checks.

Given the flexibility of qualitative anlysis, it is important to be clear and explicit about the decisions being made throughout the entire process. Braun has presented 15 criteria for conducting a good qualitative thematic analysis. These criteria focus on things like checking that each data item has been given equal attention, and checking that themes are internally coherent, consistent, and distinctive. These critera help to safeguard against the many pitfalls of thematic analysis.

As we have shown, thematic analysis is a powerful and flexible qualitative methodology. Accordingly, this thesis leverages thematic analysis to guide our study of group problem solving in introductory computational physics with the hopes of highlighting the various practices students engage in that are indicative of computational thinking. A detailed account of this process is described in Sec. 5.

Chapter 3

Context

It is important to understand the course from which we have collected our data to better understand the results of our study. That course – called Projects and Practices in Physics (P^3) – is based on a social constructivist theory of learning and a flipped/problem-based pedagogy [26]. In other words, students familiarize themselves with relevant material before coming to class, where they will work in small groups to actively and socially construct knowledge while solving complex analytical and computational physics and engineering problems. The course has intentionally been designed to encourage computational thinking wherever possible. Specifically, computational thinking has been incorporated into the notes, pre- and post-class homework, in-class feedback and assessments, and a selection of the in-class problems.

3.1 Course design

Each week in P^3, students are expected to do a number of things. They must complete the pre-class homework which is based on information that they should gather from the pre-class notes. They must then work in small groups (usually between three and four members) on two related analytical problems or a mixture of one related analytical and one related computational. These problems are delivered during the two two-hour weekly meetings (See Fig. 3.3). For the computational problem, that means reading and interpreting pre-written

31

code (i.e., a minimally working program) while they design, assess, and construct a computational force model. The small group is facilitated by either a course instructor, graduate teaching assistant, or undergraduate learning assistant who will ask relevant and pertinent follow-up questions. There are also post-class homework questions based on information gathered from the pre-class notes and the in-class problems that are due at the end of the week. This all occurs while students simultaneously prepare for the following week.

3.2 VPython

Given that the vast majority of students enter P^3 with little to no prior programming experience, we need to ensure that they are prepared to handle computational problems early in the semester. One way that we can ensure this is by requiring students to engage with the fundamental programming ideas (e.g., iteration through a while loop control structure or pre-defined mathematical functions) before coming to class through pre-class homework and notes. These notes and homework questions highlight the fundamental physical and programming ideas specific to VPython and the computational problems that will be delivered in class.

For example, consider the portion of the course notes shown in Fig. 3.1. These notes are made available to the students at the beginning of the semester and are meant to provide students with a basic understanding of the utility of VPython along with a list of common errors that novice programmers must frequently deal with. These notes provide not only a description of the error, but also a procedure for removing it while students are troubleshooting and debugging in-class code. Troubleshooting and debugging are two of the problem solving practices indicative of computational thinking that we focused our analysis

32

on.
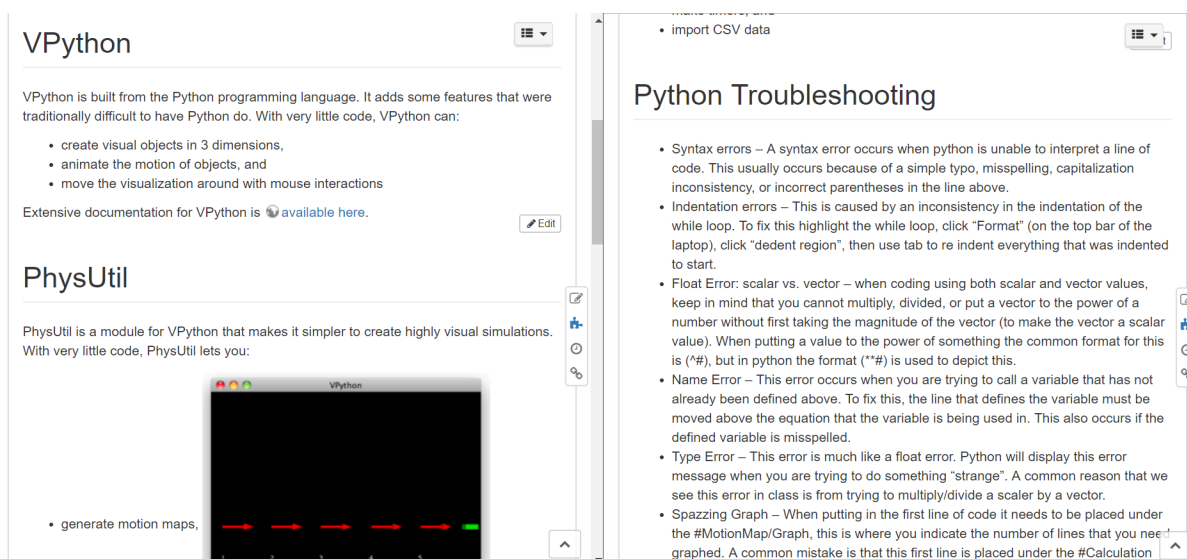
Figure 3.1: Portion of on-line notes that is made available to the students during the first week of the course. These notes introduce the fundamenatal programming ideas and a list of common errors with tips and tricks.

3.3   Pre-class work

There are other weekly notes, made available to the students at the beginning of every week, focusing more on the fundamental physical ideas that will be used during class. For example, during the third week the notes focus on uniform circular motion (most heavily used during the week's analytical problem) and the Newtonian gravitational force (most heavily used during the week's computational problem).

Aside from notes, material is also delivered to the students through weekly pre-class homework questions. Consider the pre-class homework questions shown in Fig. 3.2 that are made available at the beginning of the third week of the course. This question is meant to demonstrate that there are multiple correct ways that a unit vector can be constructed in code. Given the nature of the corresponding week's computational problem (see Sec. 3.4),

33

we expect students to be able to draw on and take advantage of this knowledge when faced with a related albeit more complicated problem. That is, we expect students to be choosing between competing solutions. Choosing between competing solutions is a problem solving practice indicative of computational thinking that we focused our analysis on.

Figure 3.2: Pre-class homework question focusing on the different ways that the magnitude of a vector can be constructed in VPython code: explicitly coding the square root of the sum of the squares of the components and using the pre-defined Python "magnitude" function.

Targeted pre-class homework questions were also developed to help students overcome challenges based on the task analysis. For example, the pre-class homework questions shown in Fig. 3.2 were developed to facilitate student understanding of the unit vector of a separation vector between two objects prior to working on the related computational problem. Given that students must grapple with using a separation vector to construct a unit vector in code during the week, these questions help to place them in the Zone of Proximal Development (ZPD) [42]. Constructing computational models is (unsurprisingly) a computational

34

modeling practice indicative of computational thinking that permeates our data.

3.4 In-class work

There are a number of in-class computational problems spread out throughout the semester (see Fig. 3.3). The first few computational problems focus on different force models (i.e., no force, a constant force, a non-constant force) and the resulting linear motion of objects. The last few computational problems focus on extended objects and their rotation. While solving these problems, groups are expected to engage in a number of computational practices that the problems have been designed around:

P1. developing and using models,

P2. planning and carrying out investigations,

P3. analyzing and interpreting data,

P4. using mathematics and computational thinking,

P5. constructing explanations,

P6. engaging in argument from evidence.

P7. and obtaining, evaluating, and communicating information.

One of the scientific practices used heavily on both analytic and computation days is that of (P1) developing and using models. Whether those models be mathematical or computational, we expect students to not only work together in groups to develop the model, but also to utilize that model in further investigations. This type of scientific practice (P1) and

35

	M	T	W	R	F	S	S
W1							
W2							
W3							
W4							
W5							
W6							
W7							
W8							
W9							
W10							
W11							
W12							
W13							
W14							

Figure 3.3: A schedule for the semester focusing on topics covered, homework/reading deadlines, and in-class problems. **Should this figure be on a landscape page?**

its associated learning goals [26] were further used to generate the in-class project that this thesis focuses on.

3.4.1 Analytic problem

In the third week of the course, students are asked to analyze the motion of a satellite orbiting Earth both analytically and computationally. For the analytic day, the groups were asked to solve for the magnitude of the velocity and radius needed by a satellite to be held in a geostationary orbit. This involves identifying two relevant equations in two unknowns and combining them to solve for the desired radius and magnitude of velocity. The information gathered during this problem can be used in the following computational problem, and the group facilitators are often observed referencing this information.

3.4.2 Computational problem

This thesis focuses on the third and most complicated computational problem delivered to the students, shown in both Figs. 3.3 and 3.4. Given its complexity, we developed a framework to help guide and ground our analysis. This framework was constructed with the help of a task analysis (see Sec. 2) of the problem. Ultimately, students must design, construct, and asses a computational model for the Newtonian gravitational force acting on a satellite in geostationary and other more general orbits.
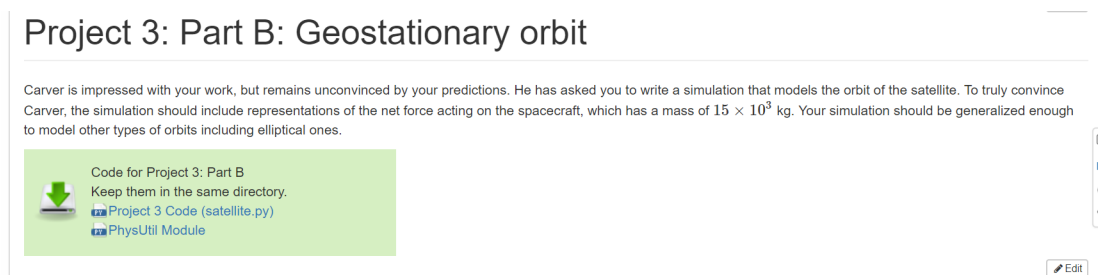
Figure 3.4: The Newtonian gravitational force problem statement delivered to the students in the third week of class.

Once the correct force has been correctly coded, the group must also grapple with adding in a visualization of a vector representing the force that they have just added. This type of motion diagram is meant to show that the gravitational force vector resulting in the orbit always points radially inward (toward the Earth). This task requires students to program as well as allows them to more easily check their conceptual understanding. Using computational models to understand a concept is a computational modeling and simulation practice that is indicative of computation thinking.

Additionally, in order to check that their model can produce a geostationary orbit, groups are asked to generate a graph showing the magnitude of the separation between the satellite and the center of the Earth vs. time. This allows them to check for a constant distance which implied a circular orbit. This task is meant, among other things, to encourage students to

visualize data, another computational practice indicative of computational thinking.

3.4.2.1 Minimally working programs

While beginning the problem, the group will observe a Minimally Working Program (MWP) similar to those seen in the two previous computational problems. This MWP has all of the structure of the code correct (the while/calculation loop and the Euler-Cromer integration) but is missing the computational force acting on the satellite (along with some inaccurate numerical values). The initial MWP code with its initial visualization are shown in Fig. 3.5.


```
1   #Window setup
2   scene.range=7E7
3
4   #Objects
5   Earth = sphere(pos=vector(0,0,0), radius=6.4e6)
6   Satellite = sphere(pos=vector(7*Earth.radius, 0,0), radius=1e6)
7
8   #Parameters and Initial Conditions
9   mSatellite = 1
10  pSatellite = vector(0,5000,0)
11
12  #Time and time step
13  t = 0
14  tf = 60*60*24
15  dt = 1
16
17  #Calculation Loop
18  while t < tf:
19      rate(10000)
20
21      Fnet = vector(0,0,0)
22
23      pSatellite = pSatellite + Fnet*dt
24      Satellite.pos = Satellite.pos + (pSatellite/mSatellite)*dt
25
26      t = t + dt
27
28      #Earth Rotation (IGNORE)
29      theta = 7.29e-5*dt
30      Earth.rotate(angle=theta, axis=vector(0,0,1), origin=Earth.pos)
```

Figure 3.5: The initial code and visualization of the MWP that is given to the students in the third week of the course.

Thus, the main task of the group is to construct a physically correct force model in code. Secondarily, they must modify numerical values to reflect the phenomenon being modeled. Ideally, this force model will be of a Newtonian gravitational form (i.e., $F_G \sim 1/r^2$) with a direction coded in terms of a separation vector (i.e., $\hat{F}_G \sim \vec{r}/r$). However, there are many other ways to go about this, and we do frequently observe groups working with other models (e.g., a centripetal force).

3.4.2.2 Tutor questions

There are a number of pre-written tutor questions as well as many on-the-fly questions generated by the tutors while in class. These questions are meant to check the students for conceptual understanding as well as to direct students toward the correct solution. For example, the tutor questions shown in Fig. 3.6 are meant to ensure that the model the group has constructed is actually general enough to generate all types of elliptical orbits given various initial conditions.

Figure 3.6: A seclection of tutor questions that focus on the computational model each group has constructed.

On the other hand, a tutor interaction like the one shown below that happens on-the-fly might encourage students to use a more general force rather than a more restricted one:

TA: you guys wanna talk about what your trategy is at hte moment

SB: i dont think we know

SA: we just, we need to figure out how to get the velocity of the spacecraft correct as well as the force net correct and then it should be fine

TA: yeah, my request, can i point in your program thats what you have for F net now [constant components] my request is to use a completely different strategy where that formula [points to Gmm/r2 on the board] is in for Fnet

SC: yeah we tried to make that yeah

SA: can we just put the number in?

TA: umm in principle you could, but id really rather you not have you do it i would like the program to be able to respond if the satellite is father away the force would be less, if the satellite is closer the force would be more so i would like it to be a dynamic program and not one that always have a fixed force

In this on-the-fly interaction, the question of weather or not their computational model will be able to handle all types of orbits is enough to indicate that the group needs to switch their model up. In this way, the tutor is able to make sure the groups stay on the desired path without directly telling them exactly what to do.

3.4.3 Feedback/Assessment

The groups are assessed on many levels in P^3. One of the most important forms of assessment is given weekly, in the form of written feedback and a numerical score. The written feedback

is based on the observed in-class performance and is designed to point out deficiencies and suggests ways to improve. The numerical scoring is based on performance in three categories: group understanding, group focus, and individual understanding.

Often the written feedback pertains to group activity with the computer. For example, the portion of written feedback shown in Fig. 3.7 is encouraging a student to allow other group members to do some of the typing. This could be requested for any number of reasons – most likely, though, because the students with less prior programming experience are not being given a chance to participate.

Feedback	Group Understanding	Group Focus	Individual Understanding
Doug, first and foremost let me say good job on working through a very difficult problem on Thursday. If you remember last feedback, we had hoped to see you playing more of an overseen role with the Vpython. Although we definitely saw more group involvement, not many other hands were doing the typing. It is going to be important that others have a chance at typing! For the future, try to use your familiarity with the computer to play more of a guiding role. As a post script, this will be your last feedback before our first exam. A few tips for success: it might be a good idea to have a designated scribe to make sure things are being written down in an organized and coherent manner, also don't forget to plan what you are doing! Take a few minutes to organize thoughts and think things through before you hastily jump into a solution. Good luck!	3.25	3.5	3.25

Figure 3.7: A snippet of written feedback given to a student after the third week.

In this way, instructors can encourage their groups to share the programming load. While doing the typing, it is very difficult to follow along without knowing exactly what is going on. This helps to engage all of the students with the material.

3.5 Post-class work

There are a number of post-class homework questions that are meant to reinforce the physics and computational concepts seen in class. During the third week of the course, these ques-

tions focus mostly on the Newtonian gravitational force. However, the post-class homework question shown in Fig. 3.8 that is delivered in the third week focuses on the previous week's computational problem (i.e., it involves a local gravitational force as opposed to a Newtonian gravitational force). Nevertheless, this post-class question involves the same Euler-Cromer style of numerical integration as seen in all computational problems. The students are expected to use the error message in order to identify an error in the code.

```
Traceback (most recent call last):
  File "ModelCar.py", line 16, in <module>
    car.pos = car.pos + vcar*dt
TypeError: unsupported operand type(s) for +: 'vector' and
'float'
```

The program as written appears below.

```
from visual import *

car = box(pos=vector(-120,0,0), size=(4.7,1.9,1),
color=color.red)
ground = box(pos=vector(0,-1,0), size=(300,1,1),
color=color.green)

mcar = 1050
vcar = 8.65

t = 0
dt = 0.01

while t < 0.6:

    rate(150)

    car.pos = car.pos + vcar*dt
    t = t + dt
```

Identify the error(s) in your program, indicate which line(s) should be changed, and write the line(s) that should be changed below:

Figure 3.8: A portion of a post-class homework question delivered in the third week of the course. This question requires students to troubleshoot and debug the code.

This type of problem helps to encourage students to identify, isolate, reproduce, and correct unexpected problems that arise while constructing computational models. Ideally, it requires students to interpret the names given to the variables being used and verify that they are defined in a correct form. Correct form means following one of the basic rules of

algebra – you cannot add a scalar and a vector.

Chapter 4

Motivation

Aside from a general interest in introductory computational physics, it is important to understand the underlying motivation(s) for this thesis. Sections from the following chapter, detailing some of those motivations, were published in the proceedings of the 2015 Physics Education Research Conference [1], and is presented here with minor modifications from its appearance in publication. It was published with second and third authors Paul W. Irving and Marcos D. Caballero, respectively.

The process of identifying an interesting computational practice, described in Sec. 4.1, was the earliest motivation for this study. We found that it was extremely difficult to define and identify the particular practice of what we named "physics debugging." Not only did the practice need to be clearly defined, it also needed to be clearly identified in the data. This required a lot of in-depth qualitative analysis and inter-rater reliability, motivating our use of the Weintrop framework and the qualitative methods of Clarke et. al.

Additionally, as described in Sec. 4.2, we found that it was very difficult to understand the qualitatively different ways in which students experienced computational introductory physics. This difficulty motivated a task analysis with a focus on identifying practices that the students were engaging in through in-class observation, as opposed to their experiences through out-of-class interviews.
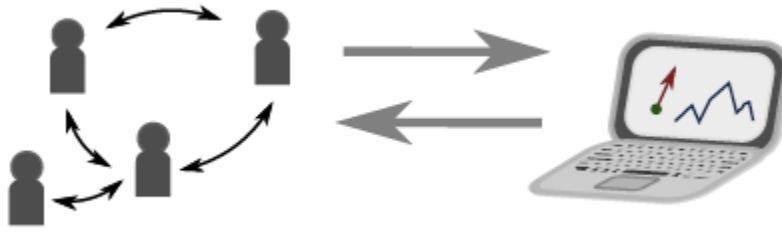
Figure 4.1: Caption.

4.1 Debugging

In this section, we present a case study of a group of students immersed in this P^3 environment solving a computational problem. This problem requires the translation of a number of fundamental physics principles into computer code. Our analysis consists of qualitative observations in an attempt to describe, rather than generalize, the computational interactions, debugging strategies, and learning opportunities unique to this novel environment.

We focus this case study on the interactions between group and computer, illustrated in Fig. 4.1, to begin to understand the ways in which computation can influence learning. Particularly, we are interested in the interactions occurring simultaneously with social exchanges of fundamental physics principles (FPPs) specific to the present task (e.g., discussing $d\mathbf{r} = \mathbf{v}\,dt$ on a motion task) and the display of desirable problem solving strategies (e.g., divide-and-conquer). These group-computer interactions vary in form, from the more active process of sifting through lines of code, to the more passive process of observing a three-dimensional visual display.

One previously defined computational interaction that reinforces desirable strategies, borrowing from computer science education research, is the process of debugging [20]. Computer science defines debugging as a process that comes after testing *syntactically* correct code where programmers "find out exactly where the error is and how to fix it. [30]" Given

the generic nature of the application of computation in computer science environments (e.g., data sorting, poker statistics, or "Hello, World!" tasks), we expect to see unique strategies specific to a computational *physics* environment. Thus, we extend this notion of computer science debugging into a physics context to help uncover the strategies employed while groups of students debug *fundamentally* correct code that produces unexpected physical results.

4.1.1 Analysis

In Fall 2014, P^3 was run at Michigan State University in the Physics Department. It was this first semester where we collected *in situ* data using three sets of video camera, microphone, and laptop with screencasting software to document three different groups each week. From the subset of this data containing computational problems, we *purposefully sampled* a particularly interesting group in terms of their computational interactions, as identified by their instructor. That is, we chose our case study not based on generalizability, but rather on the group's receptive and engaging nature with the project as an *extreme case*.[21]

The project that the selected group worked on for this study consists of creating a computational model to simulate the geosynchronous orbit of a satellite around Earth. In order to generate a simulation that produced the desired output, the group had to incorporate a position dependent Newtonian gravitational force and the update of momentum, using realistic numerical values. The appropriate numerical values are Googleable, though instructors encouraged groups to solve for them analytically.

This study focuses on one group in the fourth week of class (the fourth computational problem seen) consisting of four individuals: Students A, B, C, and D. The group had primary interaction with one assigned instructor. Broadly, we see a 50/50 split on gender, with one ESL international student. Student A had the most programming experience out
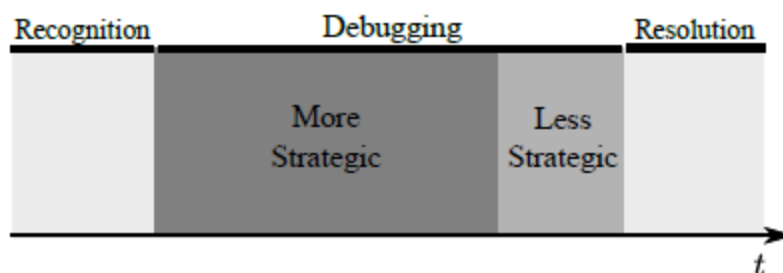
Figure 4.2: The debugging process necessarily corresponds to a phase beset on either side by the phases of recognition and resolution. Note the absence of a vertical scale, as the vertical separation merely acts to distinguish phases.

of the group. It is through the audiovisual and screencast documentation of this group's interaction with each other and with the technology available that we began our analysis.

To focus in on the group's successful physics debugging occurring over the 2 h class period, we needed to identify phases in time when the group had recognized and resolved a physics bug. These two phases in time, *bug recognition* and *bug resolution* are the necessary limits on either side of the process of *physics debugging*, as represented in Fig. 4.2. We identified these two bounding phases at around $60(5)$ min into the problem, and further examined the process of debugging in-between. That is, we focused on the crucial moments surrounding the final modifications that took the code from producing unexpected output to expected output.

4.1.1.1 Recognition

At around 55 min into the problem, following an intervention from their instructor, the group began to indicate that they were at an impasse:

SB: We're stuck.

SD: Yeah...

The simulation clearly displayed the trajectory of the satellite falling into the Earth not the geostationary or bit they expected as observed on the screencast. This impasse was matched with an indication that they believed the FPPs necessary to model this real world phenomenon were incorporated successfully into the code:

SB: And it's gonna be something really dumb too.

SA: That's the thing like, I don't think it's a problem with our understanding of physics, it's a problem with our understanding of Python.

Instead of attributing the unexpected output with a mistake in their understanding or encoding of FPPs, they instead seemed to place blame on the computational aspect of the task.

During this initial phase, we see a clear indication that the group has recognized a bug – there is an unidentified error in the code, which must be found and fixed:

SA: I don't know what needs to change here...

SD: I mean, that means we could have like anything wrong really.

Although they have identified the existence of the bug, they still are not sure how to fix it – this necessitates the process of debugging.

4.1.1.2 Physics debugging

Within the previously identified phase of bug recognition, the group developed a clear and primary task: figure out exactly how to remove the bug. Eventually, following a little off-topic discussion, the group accepted that in order to produce a simulation that generates the correct output, they must once again delve into the code to check every line:

48

SA: ...I'm just trying to break it down as much as possible so that we can find any mistakes.

In this way, the group began to not only deter mined the correctness of lines of code that have been added/modified, but also began to examine the relation ships between those lines.

For example, the group began by confirming the correctness of the form of one such line of code:

SA: Final momentum equals initial momentum plus net force times delta t. True?

SC: Yeah...

SB: Yes.

SA: O.K. That's exactly what we have here. So this is not the problem. This is right.

SD: Yeah.

That is, Student A i) read aloud and wrote down the line of code $\vec{p}_f = \vec{p}_i + \vec{F}_{\text{net}} * dt$ while the entire group confirmed on its correct form. This written line was then boxed, and was shortly followed up (ii) with a similar confirmation of the line $\vec{r}_f = \vec{r}_i + \vec{v} * dt$ that immediately prompted (iii) the confirmation of $\vec{v} = \vec{p}/m$. Thus, not only do we see the group determining the correctness of added/modified lines of code as in (i)—(iii), we further see confirmation with the links between those lines. The confirmation of the link between the lines of code (i) and (ii), representing the incremental update of position and momentum in time, respectively, was evidenced not through the mere addition of the linking equation (iii) to the list of lines added, but further through the gestures exhibited by student A. Pointing at (iii), the $\vec{v}$ in (ii), and the $\vec{p}_f$ in (i), demonstrated that the group understood that without this linking equation (iii), the velocity used in (ii) would not reflect the time updated velocity by means of (i).

The group ran through these types of confirmations with FPPs rapidly over the span of a few minutes. Once the group had confirmed all the added/modified lines of code to their satisfaction, the discussion quieted down. The FPPs were winnowed from the discussion, and after a little more off-topic discussion we find them seeking help from the instructor:

SD: Maybe we should just stare at him until he comes help us...

Suddenly, a haphazard change to the code:

SA: You know what, I'm gonna try something...

where Student A changed the order of magnitude of the initial momentum a few times. This modification even tually resulted in a simulation that produced the correct output.

4.1.1.3 Resolution

At about 65 min into the problem, Student A changed the order of magnitude of the momentum one final time, which produced something closer to the output that they expected:

SA: Oh wait... Oh god...

SD: Is it working?

The satellite now elliptically orbited the Earth. This marks the end of the debugging phase and the beginning of the resolution phase the bug had successfully been found and remedied. Given that the only line of code modified to produce this change was the initial momentum, they began to rethink the problem:

SD: I think that is the issue is that we don't have the initial momentum...

SA: ...momentum correct?

That is to say, the group pursued the issue of determining the correct initial momentum with the added insight gained through debugging fundamentally correct VPython code.

4.1.1.4 Strategies

This case study has described two strategies (one more and one less strategic) employed by a group of students in a physics course where students develop computational models using VPython while negotiating meaning of fundamental physics principles. These strategies arose through the group's process of debugging a fundamentally correct program that modeled a geostationary orbit. The additional data we have collected around students' use of computation is rich, and further research is needed to advance the depth and breadth of our understanding of the myriad of ways in which students might debug computational models in physics courses.

4.2   Phenomenography

We also conducted a phenomenography in order to characterize the qualitatively different ways in which students were experiencing the problem. In order fill the outcome space, we looked for the variation in students descriptions of what we called the "critical" components of the problem. This was accomplished through post-class interviews where we followed a semi-structured protocol that was developed specifically for this case study. Some of the very interesting results that can be generated from this type of analysis is presented in [23].

Chapter 5

Observations

Throughout our analysis in this thesis, we have made many different types of observations, and have used those observations to help answer our research questions (for example, see Sec. 2.5). Accordingly, it is important that we take some time to elaborate on the process of and results from those observations. More specifically, in this chapter, we detail the method of our analysis (i.e., the data reduction, the coding process, and the inter-rater reliability) and illustrate the identification of some of the most frequent practices (e.g., troubleshooting and debugging, assessing computational models, and creating computational abstractions).

5.1   Analysis

Our full analysis involves different stages: first, the initial data was collected and subsequently reduced in order to provide a manageable set of data; next, an independent coding scheme was generated – using the Weintrop framework from Sec. 2.1 – to help identify computational practices; and finally, multiple inter-raters were used to ensure the reliability of the analysis. Each of these three stages are detailed below.

5.1.1 Data reduction

Our total set or corpus of data consists of in-class video of nine groups of four individuals working. Each group works on three computational problems (twenty-seven videos in total) that increase in difficulty/complexity as the semester progresses. These computational problems, described in Sec. 3.3, require students to construct various computational force models in code. Each week, the appropriate force model increases in complexity and generality. Specifically, the first problem involves a constant zero force, the second problem involves a constant non-zero force, and the third problem involves a non-constant force.

In order to first reduce the corpus of our data to a more focused and manageable set, we payed attention to when students were making the most progress toward a solution. The frequency of independent progress being made increased as the complexity of the problem increased (i.e., students made the most independent progress on constructing the Newtonian gravitational force model). Here, we are defining "independent progress" as progress that is ultimately made by the group without any instructor intervention. We believe this is due, in part, to their lack of prior programming experience coming into the course – in other words, they progress quickly. For example, on the first problem, many groups struggled with a basic calculation (while) loop. By the time they see the third problem, they have already gained a little experience and are becoming accustomed to the norms of the course (e.g., their familiarity with the format, the programming environment, etc.).

Our initially reduced set of data consists of transcripts from in-class video (both side-view and overhead-view) of nine groups working on the Newtonian gravitational force problem from Sec. 3.4. We also collected computer screencasts to capture exactly what students are doing when they type/click on their group laptop. Following the suggestions of thematic

	SA	SB	SC	SD	TA
365			thats close to...		
366	nine point is meteres per second though...				
367			[looks in notes]	[looks in notes]	
368				yeah its that	
369				gravity equals to like gravity, of gravity equals F net	
370				equals to gravity equals to {writes Newtonian force on board}	
371		whats that?			
372				thats the constant of	
373		oh the G yeah			
374		six point six...			

Figure 5.1: A portion of transcript meant to highlight the indication of unspoken and inferred actions. For example, line 367 shows this group looking in their notes for an equation. The equation that they find is written down in line 370.

analysis (see Sec. 2.5), we began with a full transcription of the in-class video to the best of our abilities. Any inaudible sections are indicated, with long pauses being indicated by ellipses (...). To distinguish between unspoken actions (e.g., pointing to an equation) and inferences made by the primary researcher (e.g., a group referring to a previously used equation), we follow the convention of square brackets ([]) and curly brackets ({ }), respectively. For example, Fig. 5.1 shows a portion of transcript highlighting these various indications.

Once we had reduced our data corpus to a more manageable and focused set of nine transcripts, we continued our investigation into the computational practices students were engaging in. Each transcript was read multiple times in order to generate a low-resolution but coherent picture of what each group was doing – or at least, what each group was trying to do. This type of "familiarization" with the data is a crucial step as outline by Braun et. al. Ultimately, this low-resolution picture helped us to identify the off-topic and otherwise

irrelevant discussion in order to remove those portions of the data from our analysis.

More specifically, each transcript was initially analyzed with an eye towards identifying discussion where students were i) solving the satellite problem and ii) using a computer (e.g., typing or reading through lines of a program). All other discussion then could be considered off-topic and safely discarded. For example, groups are often seen discussing homework for other classes that in no way relates to the Newtonian problem. Similarly, groups can often be seen discussing recent social events (e.g., a concert). This type of off-topic and otherwise irrelevant discussion, although important for the social cohesion of the group, can safely be discarded. In this way, we further reduce our data set by about one quarter. With each of nine transcript being about fifteen-hundred lines of speech/action, this translates to about ten-thousand lines of on-topic discussion for further analysis.

A closer analysis of this on-topic discussion is where we begin to more clearly define what computational practices look line within our data. This closer analysis started with the search for a number of characteristics (as described in Sec. 2.1), within the on-topic discussion. For example, the key characteristics for the practice of troubleshooting and debugging are: i) to identify and isolate an unexpected error, ii) articulate how to reproduce the error, and iii) work to systematically correct it. These characteristics, once identified, can be used to justify the classification of an excerpt as the computational practice of troubleshooting and debugging (recall that each computational practice may be indicative of the computational thinking as described in Sec. 2.1). This justification allows us to define the computational practices we see in our data. A detailed account of this process of justification is described below, with applications to specific examples following in Sec. 5.2.

5.1.2 Coding process

In order to justify the classification of an excerpt as a particular computational practice, we started by systematically coding our data. This systematic coding process was applied to three streams of data: the side-view video, over-head video, and computer screencasts. These three streams were then used to generate three types of rationale: rationale accordingly to the framework, rationale within an individual excerpt, and rationale beyond an individual excerpt. These three types of rationale are described in detail below.

In terms of the framework, we identified the various characteristics that manifested themselves in the actions and speech of each group and compared them to the Weintrop framework. Each practice, according to the framework, has any number (between one and seven) of related characteristics. The more related characteristics that we see in an excerpt, the more confident we are in classifying that excerpt as a particular practice. For example, "identifying an unexpected error in code" is one of the required characteristic of troubleshooting and debugging. Similarly, "working to systematically rectify the unexpected error" is clearly a related but distinct characteristic. The identification of either of these characteristics individually would be hinting at the practice of troubleshooting and debugging, but both of them simultaneously makes a stronger claim. This type of rationale can be found in Column G of Fig. 5.2.

Within an individual excerpt, we are able to focus in on what each member of the group says and does as they work toward a clear and focused goal. Any rationale of this type usually references line numbers pertaining to specific lines of speech/action within the excerpt that embodies the characteristic in question. In this way, we closely tie our rationale and the framework to the data. For example, a group might identify an unexpected error in their

program and say:

SC: *(756)* oh there it is {the error message}

SB: *(757)* where?

SC: *(758)* in the thing {shell} on the screen...

In this exchange, Student C has found the error message from the shell buried under a few other windows. This error message is ultimately used by the group to track down the cause of the unexpected error. In this way, we clearly see a group working to identify an unexpected error in our data. This type of rationale can be found in Column H of Fig. 5.2.

Beyond each individual excerpt (i.e., looking at each transcript as a whole), we are able to generate a low-resolution picture that captures the overarching goals that each group is working toward. This low-resolution picture helps us to contextualize each individual excerpt within the broader transcript. There are many ways to contextualize a particular excerpt of data (e.g., in the context of the group, the classroom, the university, the state, etc.), and relating it to other excerpts is one of the most important. For example, within an individual excerpt, a group might reference – without defining – an equation:

SA: *(894)* should we try **that one** equation?

SB: *(895)* yeah i think we should do that...

SA: *(896)* okay

SC: *(897)* yeah thats a good idea lets use that one

Using our low-resolution picture of the transcript as a whole, we can track back through time (often minutes, sometimes longer) to find out exactly what vague equation they are referencing:

	A	B	C	D	E	F	G	H	I
		Excerpt #							
	SA	SB	SC	SD	TA	Tags	Framework	Within	Without
Line #	Student A speech and action.	Student B speech and action.	Student B speech and action.	Student D speech and action.	TA speech and action.	Here we classify the excerpt as a particular practice according to the framework.	Rationale according to the framework goes here. This includes language frrom the defintions according to Weintrop and the language used in the other two levels of rationale.	Rational within the excerpt goes here. It usually references a line #.	Rationale beyond the excerpt goes here. It usually refences another Excerpt #.

Figure 5.2: The template used for the coding process. Each excerpt is numbered, each line of speech/action is numbered and attributed to an individual member of the group, and the three types of rationale are used to justify the classification of a particular practice.

SC: *(120)* how about we use **the equation**...

SC: *(121)* [writes G m M over r squared]

SC: *(122)* ...and then multiplied by r hat

SD: *(123)* i dunno...

Any rationale provided at this level usually references the number of another excerpt that provides the necessary additional information. This level of rationale can be found in Column I of Fig. 5.2.

This coding process was followed for nine groups to generate about five-hundred candidate excerpts, each excerpt having multiple practices, and each practice having the three types of rationale described above. Each excerpt has anywhere from one to four possible practices identified with supporting rationale. That equates to roughly three-thousand individual justifications that must be found within our data.

The three types of rationale described above, though not necessarily persuasive individually, when taken together can provide a reasonable justification for the classification of an excerpt as belonging to a particular computational practice: the rationale from the framework provides incomplete but guiding definitions, the rationale within an individual excerpt ties us closely to the data and the immediate actions that a group is taking, and the rationale

58

beyond an individual excerpt helps to contextualize those immediate actions and speech.

5.1.3 Inter-rater reliability

In order to ensure not only reasonable, but also *reliable* justifications for the classification of the various computational practices within our data, we followed an iterative process of inter-rater reliability. One primary researcher was joined by three impartial inter-raters, ensuring a robust coding process and stronger claims through iterative critique and discussion.

Initially, the data was coded by the primary researcher, relying heavily on the Weintrop framework and the qualitative methods described in Ch. 2, to generate an initial set of rationale for each candidate excerpt. This initial set of rationale for a particular excerpt, consisting of the three types of rationale described in the section above, was then taken as a whole to formulate an initial level of confidence: low, medium, or high. Low confidence was usually given to excerpts containing only a few of the characteristics needed by a practice, or to excerpts where the identification of an individual characteristic was in serious question. Medium confidence was given to excerpts containing most of the characteristics required by a practice, or to excerpts where the identification of individual characteristics was probable. High confidence was given to excerpts containing all of the required characteristics for a practice, or to excerpts where the identification of each individual characteristic was self-evident. Examples of excerpts belonging to these different levels of confidence are shown in Fig. 5.3.

A subset of the data containing a variety of computational practices and levels of confidence was then shared with multiple inter-raters. Each inter-rater subsequently tested the strength of our initial claims through discussion by asking questions and making suggestions. These suggestions, once mutually agreed upon, were incorporated into the rationale.

Inter-Rater Comments	Tag	Rationale from framework
I actually think this is an example of absrtraction	Computational abstraction	The group is identifying, creating, and using a computational abstraction as they work toward a goal.

Inter-Rater Comments	Tag	Rationale from framework
I think I am struggling with what is meant by levels here. I see them trying to write a constant, but I don't see the larger connections	Thinking in levels	The group has identified different **levels** in a system.

Inter-Rater Comments	Tag	Rationale from framework
what is abstraction? I'm not seeing it, but maybe I'm using some colloqial lens that is inappropriate	Creating computational abstractions	The group has **identified** a computational **abstraction** as they advance.

Figure 5.3: Examples of the three levels of confidence are shown in green (high), medium (yellow), and low (red) according to the supporting rationale from the framework. Each inter-rater suggestion is used to modify or solidify the level of confidence given to a particular practice.

Figure 5.4: The initial rationale generated for an excerpt along with inter-rater suggestions and subsequent modification. With the addition of some requested information, the strength of the rationale was improved and the confidence was promoted from medium to high.

For example, Fig. 5.4 shows one inter-rater asking a clarification question as to what the verbatim output of the shell in a particular excerpt was. The answer to this clarification question, though no obvious given the initial rationale, proves to be relevant and necessary to the strength of the rationale. Accordingly, this inter-rater suggests that this additional information be added to the rationale to improve confidence. This process of generating reliability through asking questions and making suggestions was followed iteratively to further strengthen each claim.

5.2 Computational practices

By analyzing all of the data with the methods described above, we have identified a number of practices that show up in our data. These practices and their frequencies within our data

Data	65
Collecting data	0
Creating data	19
Manipulating data	7
Analyzing data	31
Visualizing data	8
Modeling and simulation	97
Designing computational models	32
Constructing computational models	18
Assessing computational models	27
Using computational models to find and test solutions	7
Using computational models to understand a concept	13
Computational problem solving	66
Preparing problems for computational solutions	2
Choosing effective computational tools	0
Assessing different approaches/solutions to a problem	9
Creating computational abstractions	13
Developing modular computational solutions	10
Programming	21
Troubleshooting and debugging	11
Systems thinking	44
Defining systems and managing compelxity	6
Investigating a complex system as a whole	7
Understanding the relationships within a system	6
Thinking in levels	13
Communicating information about a system	12

Figure 5.5: The frequency of each practice that was found within our unique data set.

are summarized in Fig. 5.5. In total, we identified roughly 250 occurrences of individual practices, with some practices occurring frequently and some occurring never. The most frequent practices, though found within our data, can be expected to arise just as frequently in sufficiently similar classrooms and deserve a fair amount of attention.

The remainder of this section provides concrete examples of some of the most frequent computational practices that we found in our data. We are focusing on those practices that occur with high frequency within one group or occur with moderate frequency across multiple groups. These practices are (in no particular order): creating and analyzing data within the data practices; designing, constructing, and assessing computational models within the modeling and simulation practices; programming, creating abstractions, and troubleshooting and debugging in the computational problem solving practices; and thinking in levels and

62

communicating information within the systems thinking practices.

Although the examples that follow are meant to clearly illustrate some of the common computational practices that we have observed, they do not come without their own limitations. Although we have tried our best to as unbiased as possible, the nature of this researcher requires a fair amount of subjective interpretation. Accordingly, Ch. 6 will provide additional discussion on the caveats, limitations, and concerns associated with some of the common practices that we have focused on.

The other less frequent practices, though not the focus here, are still of research interest. Some of these less frequent but nonetheless important computational practices can be found in the Appendix.

5.2.1 Creating data

The computational practice of creating data, as defined by Weintrop et. al, involves the generation (as opposed to the collection) of computer data while "investigating phenomena that cannot be easily observed or measured or that are more theoretical in nature." This type of data creation frequently arises in physics and engineering given the infeasibility of data collection in many realistic situations. For example, complex computer models can be used to generate data that can be used to optimize launch conditions for satellites and manned rockets when real-world collection of data is too costly or dangerous. The fundamental characteristics associated with this practice, as summarized in Tab. 5.1, are: i) defining a computational procedure that automatically/algorithmically creates data and ii) using that procedure or the resulting data to advance the overall goals of the task.

Consider Excerpt 9 from Group H. Over the course of two hours, this group can be seen ensuring that their MWP will dynamically update the position of the satellite. This entails

Characteristic	Qualities
Automating	The data that is being created should be done so in an automatic or algorithmic manner. For example, an Euler-Cromer style integration is frequently used to generate large sets of numerical data representing various physical phenomena in time.
Advancing	Each group should ultimately be advancing toward completion of the specified task. For example, creating an algorithm that generates the various momenta of the satellite can ultimately be used to help generate a simulation of its trajectory.

Table 5.1: The characteristics and associated qualities pertaining to the computational practice of creating data: automating the creation of data that helps to advance toward goals.

ensuring that the momentum of the satellite will also dynamically update. Accordingly, the group works to construct a computational algorithm that will automatically create sets of data representing the position and momentum of the satellite over time. These sets of data are then ultimately used to advance toward completing the goal of producing of a realistic visualization of the trajectory of the satellite.

Early on, the group can be seen discussing their goal of generating a visualization of the satellite's orbit (lines 195-196). They consider changing the initial position of the satellite (line 199) to what they calculated from the previous problem:

SD: *(195)* So, it's mostly just trying to figure out how to get it {the program} to display an orbit...

SA: *(196)* Yeah, it is.

SC: *(197)* Wait, we have to change the position, don't we?

SB: *(198)* I think the initial position stays there, we have to update position though...

SC: *(199)* Yeah, we have to change the initial position to what we found... it was this far away, you know?

SA: *(200)* Yeah.

SB: *(201)* Yeah.

SA: *(202)* Which was... four point four two times ten to the seven.

SB: *(203)* Four point two... [codes]

They make the distinction between changing the initial position of the satellite and changing the way that the position updates over time (line 198). This is an important distinction because each change involves vastly different amounts work to accomplish, and only one results in the automatic/algorithmic creation of data. That is, changing the initial position of the satellite is a simple change of a numerical value, whereas changing the way that the position updates over time involves defining a set of algorithms with multiple variables inside of the calculation loop. Ensuring that the position updates properly is a big advancement toward their goal of producing a realistic visualization.

Eventually, they propose an Euler-Cromer style algorithm to automatically update the position of the satellite (line 222) in terms of its momentum, mass, and time:

SB: *(217)* Alright...

SB: *(218)* Okay, so we have to add its new position.

SA: *(219)* But it has to update its position every time...

SB: *(220)* Right.

SA: *(221)* So we have to make it update.

SB: *(222)* Satellite position plus momentum of the satellite...

SA: *(223)* Over the mass?

SB: *(224)* Times the change in time... yeah so its, yeah.

SB: *(225)* But the momentum is always changing...

Although the group has clearly laid out the way that the position of the satellite will need to change (line 222), they have raised another concern in terms of the momentum of the satellite (line 225). In other words, they have defined a procedure to automatically calculate the positions of the satellite, but still need to define a procedure to automatically calculate the momenta.

Later, as the group works toward defining a procedure to change the momentum of the satellite over time, they recall the concept of both iterative prediction (line 684) and Newton's second law (line 695) from the notes:

SB: *(681)* So we gotta figure out how to change the momentum in there {the code}.

SB: *(682)* What was the equation from last week?

SA: *(683)* Umm... F grav... no.

SD: *(684)* What about using iterative prediction for like future positions?

SD: *(686)* Right?

SC: *(687)* The change in momentum would be the net force times...

SB: *(688)* Because the force is mass times acceleration...

SC: *(689)* That would be it, yeah.

SB: *(690)* So integrate that.

SC: *(692)* Changing momentum is force times change in time...

SB: *(693)* Oh, there we go, nice.

SA: *(694)* Wait what is it?

SB: *(695)* The change in momentum is the net force times change in time.

With these two algorithms defined, their MWP is ready to automatically and dynamically update the position and momentum of the satellite. Afterward, the group spends a fair amount of time incorporating the appropriate force model into their code. The construction of these algorithms, along with the correct force model, shows a clear advancement toward their goal (line 195) of generating a visualization of the satellite's orbit.

To summarize, the group can be seen *automating* the generation of sets of data representing the position and momentum of the satellite over time. Further, with these sets of data, the group is ultimately *advancing* their progress toward producing a visualization of orbital motion. Given the identification of these two characteristics, we classify this excerpt as the computational practice of creating data.

5.2.2 Analyzing data

The computational practice of analyzing data, as defined by Weintrop et. al, usually involves large sets of data (that have either been created or collected) where groups are "looking for patterns or anomalies, defining rules to categorize, and identifying trends and correlations." This type of analysis shows up frequently within the field of physics, especially given the computational nature of many (if not most) modern investigations. For example, extremely large sets of data are generated while investigating the formation and evolution of galaxies

67

Characteristic	Qualities
Analyzing	This is a broad term that usually involves at least one of many types of analysis. For example, sorting a set of data into different categories, looking for trends or patterns within a given set, looking for correlations between multiple sets, and/or identifying outliers and anomalies are all considered to be different types of analysis.
Concluding	The information (e.g., a pattern or trend) gathered from the analysis of a set of data should ultimately be used to make or draw some conclusion. This characteristic, though an important one, is not necessarily required for a group to be analyzing data.

Table 5.2: The characteristics and associated qualities pertaining to the computational practice of analyzing data: a general process of analysis leading to conclusions.

throughout the universe. Being able to effectively analyze a large set of data is a crucial skill within many interdisciplinary fields. The fundamental characteristics associated with this computational practice, as summarized in Tab. 5.2, are: i) a general process of analysis (detailed in Tab. 5.2) and ii) a conclusion being drawn based on that analysis.

Consider Excerpt 35 from Group H. Overall, this group can be seen engaging in the process of analysis of a set of data that represents the net force acting on the satellite, and drawing a conclusion based on the results of that analysis. The particular process of analysis observed in this excerpt involves both categorization and patterning. The categories that the data are placed in are: a) large-scale numbers and b) vector quantities. The trend that the group recognizes is that the set of data representing the net force is time dependent.

Prior to the beginning of this excerpt, the group adds a print statement (i.e., `print(Fnet)`) into their calculation loop to print off the numerical values (x-, y-, and z-components) of the net force acting on the satellite over time. They do this to check that their model is producing the expected values:

SD: *(1330)* How many times does this calculation loop run through?

SB: *(1331)* A lot...

SD: *(1332)* Yeah.. a lot [looking at the output].

SB: *(1333)* However many seconds are in a day.

SA: *(1334)* Eighty six thousand.

SD: *(1335)* Wow...

SB: *(1336)* Yeah doing it line by line is not gonna be easy.

With this print statement, they are creating a large set of data (line 1332) that is subsequently analyzed.

The group confirms that their print statement is displaying a large set of data that represents the net force on the satellite (line 1338). At the same time, they begin to categorize the data and look for trends:

SD: *(1337)* It's not showing it the satellite because I think the {window} scale is too small.

SD: *(1338)* But its outputting all of it the forces, and it is...

SD: *(1339)* It's changing too I think.

SB: *(1340)* How big are they?

SB: *(1341)* I'm assuming were talking about F grav...

SC: *(1342)* Yeah, it is big.

One trend that the group suggests (line 1339) is that the values in the set have some sort of time dependence. Similarly, one category that the group places the data in (line 1342) is that of having a large order of magnitude – which is expected given the type of force that they are analyzing.

Mistakenly, the group believes that the trend of time dependence that they have identified in their data is not the expected or desired one. In other words, they suggest that the set of data should be constant in time (line 1343):

SB: *(1343)* Uhh... I don't think its supposed to be changing.

SB: *(1344)* Not a good sign.

SB: *(1345)* Do we have it as a vector or a scalar right now?

SD: *(1346)* Right now we have it as a vector.

Additionally, the group further categorizes the set of data as being a collection of vectors as opposed to a collection of scalars (line 1346). This focus on the vectorial nature of the net force ultimately helps them to draw a conclusion about how it should behave as the satellite changes position.

After a little off-topic discussion, the group begins to consider how the various components of the net force should not only change in time (line 1425), but should also remain a particular size (line 1434):

SB: *(1420)* We need F grav to be a vector.

SD: *(1421)* We have it as a vector... it is a vector right now.

SB: *(1422)* How?

SA: *(1423)* How do you have it as a vector?

SD: *(1424)* I initiated it as a vector.

SB: *(1425)* Right, but it needs to move.

70

SD: *(1426)* Oh, does it have to be negative?

SB: *(1427)* Either way, it has to be in the x and the y direction...

SD: *(1428)* Oh well then you just do this [adds the force for the x-component]...

SB: *(1429)* Because... but it's the components that would make F grav bigger than we need it to be?

SD: *(1430)* Why?

SB: *(1431)* Because a component vector... if we have one like that [draws a vector toward the fourth quadrant] then it's gonna be out to there...

SC: *(1432)* No, it would be double.

SB: *(1433)* Right it would be that long.

SB: *(1434)* And we just need it to be that long.

SD: *(1435)* So just divide it by two then?

SB: *(1436)* Except it changes in time...

SB: *(1437)* Because when it's right here, it's only going down, and when it's right here it's only going across...

SB: *(1438)* But when it's right here, it's going down and across...

SC: *(1439)* Yeah.

In other words, although the net force has been initiated as a vector, it has been initiated as a constant vector (pointing only in the y-direction). The group reaches the conclusion (line

71

1437) that the force must be modified so that it can change directions depending on where the satellite is located relative to the Earth. Furthermore, they conclude that it is important that magnitude of the net force remain a constant (line 1434). These conclusions ultimately lead them to rethink their force model.

To summarize, this group can be seen *analyzing* a set of data representing the net force acting on the satellite over time. They have identified the *trend* that the data changes over time, and the data were placed in the *categories* of being large-scale numbers and being vectors quantities. The *conclusion* that the group makes is that the net force should not only be a vector, but that its components should be able to oscillate between the x- and y-component depending on where the satellite is. Given this process of analysis and the conclusions being drawn, this excerpt is thought to illustrate the computational practice of analyzing data.

5.2.3 Designing models

The computational practice of designing computational models, as defined by Weintrop et. al, involves the process of making "technological, methodological, and conceptual decisions." These types of decisions are frequently dealt with in the STEM discipline given the complexity of modern scientific endeavors. Scientific rigor and sound methodology must be maintained while using tools at the forefront of technology (i.e., computation) to investigate modern phenomena. At the same time, developing a deep conceptual understanding of the models and the phenomena that they represent is playing an increasingly important role in the sharing and communication of scientific information. Accordingly, the fundamental characteristics associated with this computational practice, as summarized in Tab. 5.3, are: i) defining the components of a model, ii) describing how the components of a model in-

Characteristic	Qualities
Defining	Each individual component of a model must be separately defined in code. For example, the mass of an object and the local acceleration due to a planet can be separately defined and used to construct the corresponding local gravitational force.
Relating	The group must describe the way that the individual components of the model relate to the phenomenon that is being studied. This relationship usually mirrors an equation or an expected type of behavior. For example, the Newtonian gravitational force follows an inverse square position-dependence.
Predicting	The group must articulate what information their model will provide them, and use that information to make predictions about the time evolution of a phenomenon given initial conditions. For example, a force model can generate the various values of the force acting on an object at different positions in time. This set of data can then be used to make predictions about the motion of the object.

Table 5.3: The characteristics and associated qualities pertaining to the computational practice of designing a computational model: defining components, relating them to one another, and using them to make predictions.

teract, and iii) articulating what predictions can be made with the model. In keeping with the recent literature on modeling in education research, we limit our investigation to models pertaining to the force acting on the satellite (e.g., a local gravitational force model or a Newtonian gravitational force model).

Consider Excerpt 11 from Group B. Throughout this excerpt, the group can be seen working to incorporate a centripetal force model (i.e., $\vec{F}_{\text{cent}} = -\frac{mv^2}{R}\langle\cos\theta, \sin\theta, 0\rangle$) into their code. Ultimately, the group is dissuaded from using this particular model through discussion with the TA. Nevertheless, this excerpt is a clear illustration of the practice of designing a computational model.

A few minutes into beginning the problem, the group has recognized that they need to use a force model (line 118) to calculate the trajectory of the satellite, as opposed to just plotting it using the expected radius (line 116):

SA: *(111)* Now were saying that it's {the radius} a variable...

SA: *(112)* So what do we want to do with this other number?

SB: *(113)* Well, you said the radius from here to here is not gonna be the same as from here to here?

SA: *(114)* Yeah.

SB: *(115)* Well, should we... could we Google how, like how much farther or shorter it is from here to here?

SD: *(116)* Okay, I think actually what it's trying to get us to say is that we can't just plot its path around by using the radius of the orbit...

SA: *(117)* Right.

SD: *(118)* We have to actually use the force that is acting on it to find it's path.

SA: *(119)* We have to use the force.

SD: *(120)* We have to use the force.

The group has begun to articulate the information that their model will provide them, even if they have not yet decided on the particular model. In other words, their force model will allow them to make predictions about the position and trajectory of the satellite.

After a little off-topic discussion, the group decides on a particular force model to use:

SD: *(152)* The force is like v squared... the force is uhh v squared times m over radius of orbit.

SD: *(153)* Correct me if im wrong...

SA: *(154)* Sorry?

SD: *(155)* The force is equal to mass times v squared over radius of the orbit.

SB: *(156)* So maybe we could just find it {the force} at that distance?

SD: *(157)* Well, we have access to a variable that represents our radius of orbit...

SB: *(158)* And we have mass.

SD: *(159)* And we have mass.

SC: *(160)* We found the velocity last time...

SB: *(161)* And we know the radius and know the velocity.

SB: *(162)* So we can just find the net force.

Here, the group is clearly identifying the individual components of the centripetal force model (lines 157-161) and making sure that they are separately defined in code. Additionally, they have identified a clear mathematical relationship between them (line 152) that they recall from memory.

Before jumping into the construction of the newly proposed model, they spend a little time discussing its behavior and how it relates to the phenomenon:

SD: *(182)* If we could get it {the force} to oscillate between maximums we could get a rotation...

SD: *(183)* But how do we represent that as a force... because it's obvious that they want us to do that.

SA: *(184)* Sine and cosine?

SD: *(185)* Sine and cosine?

SA: *(186)* If we do sine and cosine, if we have both of them, one in the x, one in the y, like this [points to notes] is saying...

SA: *(187)* Then even if one goes to zero, like you were saying, then the other one is gonna be close to one.

SA: *(188)* And so...

SD: *(189)* We have to use our angles?

SC: *(190)* Ohhh...

SD: *(191)* And we have access to angles that are defined below.

SD: *(192)* Oh my god, that's so great, that's perfect, you're totally right.

Specifically, they articulate the way that the components of their force model will need to oscillate to cause a rotation (line 182). This oscillatory behavior has a direct relation to the mathematical sine and cosine functions that they plan to use (line 184) – as one component approaches a value of zero, the other component will approach a value of one (line 187). They also identify yet another individual component of their model (line 191) with the angle of the satellite.

To summarize, the group begins by recognizing that using a force model will allow them to *predict* the trajectory of the satellite in a more general way (line 118). After deciding on a centripetal force model, they then separately *define* the individual components of the mass, velocity, radius, and angle of the satellite (lines 157-161 and 191). Finally, they *relate* the sinusoidal nature of the model to the expected sinusoidal behavior of the satellite's trajectory

Characteristic	Qualities
Assuming	In designing a computational model, certain assumptions are invariably taken into account. These assumptions – regardless of how appropriate or valid – should be identified and clearly articulated by the group. For example, the assumption that the satellite will always be traveling in a perfectly circular orbit, although a poor one, is still an assumption.
Validating	As more assumptions are built into a model, its validity should continually be checked to ensure its predictive accuracy. For example, assuming that an orbiting satellite is acted on by a constant net force is not valid for long periods of time.

Table 5.4: The characteristics and associated qualities pertaining to the computational practice of assessing a computational model: identifying assumptions and validating them.

(lines 182 and 186). Given these three characteristics, this excerpt is a clear illustration of the computational practice of designing a model.

5.2.4 Assessing models

The computational practice of assessing a computational model, as defined by Weintrop et. al, involves "understanding how the model relates to the phenomenon being represented." This is a crucial step in the process of modeling – without an assessment of the validity and meaning of the results (i.e., without a deep understanding), the model is almost certainly useless. The fundamental characteristics associated with this crucial computational practice, as summarized in Tab. 5.4, are: i) identifying assumptions built into the model and ii) validating the model. These two characteristics, if confidently observed within an excerpt, would serve to classify that excerpt as the computational practice of assessing a computational model.

Consider Excerpt 9 of Group C. Generally speaking, the group can be seen working to incorporate a gravitational force into their code. Early on, they recognize that their code is missing the net force on the satellite, and subsequently spend about thirty minutes deciding

if and how they should should incorporate one. Eventually, they reach a conclusion to add a gravitational force based on their assessment of a couple of different models (i.e., a local gravitational force and a Newtonian gravitational force).

A few minutes into the problem, the group considers what happens to the initial momentum of the satellite as their program runs (line 256):

SA: *(253)* Umm...

SB: *(254)* Okay.

SA: *(255)* So that's our initial momentum.

SA: *(256)* And then what happens {to the momentum}?

SD: *(257)* And then...

SA: *(258)* We need, we have it...

SA: *(259)* The net force equation is what's wrong...

SD: *(260)* Yeah and the net force equals to like gravity, right?

Obviously the group is concerned with the state of the net force equation (line 259), and a proposal is made to set the net force equal some sort of gravitational force (line 260). This is the beginning of the assessment of their net force model.

They continue to discuss and validate the type of gravitational force that they plan to incorporate into their code. Specifically they wonder what numerical value they should be using (line 262), and they suggest using the local gravitational constant ($g = 9.81\,\mathrm{m/s^2}$):

SD: *(261)* So we just need to like plug in the value of gravity right?

SA: *(262)* Yeah... but what's the value that we need?

SA: *(263)* Because we have um... we have um... we have...

SA: *(264)* Mass in kilograms and we have the radius of orbit in kilometers, obviously we all know like nine point eight number...

SC: *(265)* That's only close to the surface of the Earth...

SA: *(266)* Nine point is meters per second though...

However, they recognizes that their satellite is not particularly close to the surface of the Earth (line 265), and that the local gravitational constant is not particularly valid at the actual distance. In other words, the group can be seen validating their computational model based on the particular situation.

Eventually, the group does decide on a particular gravitational force to use (line 270):

SC: *(267)* [looks in notes]

SD: *(268)* Yeah it's that [points to equation].

SD: *(269)* Gravity equals to like gravity, of gravity equals F net...

SD: *(270)* Equals to gravity equals to [writes Newtonian force on board]...

SB: *(271)* What's that?

SD: *(272)* That's the constant of...

SB: *(273)* Oh the G yeah.

SB: *(274)* Six point six...

This force involves the universal gravitational constant ($G = 6.61 \times 10^{-11}\,\mathrm{N\,m^2/kg^2}$) as opposed to the local gravitational constant, which they clearly state (line 273). Again, the group has ensured the validity of their net force model by assessing the location of the satellite and subsequently using the appropriate gravitational constant.

Before getting to far, the group takes some time to clearly articulate an assumption (line 275) built into their model:

SA: *(275)* Sorry i just wanted to write here that we're making an assumption [writes on WB].

SD: *(276)* Yes.

SD: *(277)* F net equals to gravity.

SD: *(278)* Yes.

SD: *(279)* Equals to...

SA: *(280)* I just did that [adding an E] to show that that's of the Earth.

SA: *(281)* Does everyone agree that this is an assumption?

SC: *(282)* Yeah.

The fact that the only force acting on the satellite is a gravitational force is really just an assumption (although a good one) made at this point. The group specifically takes the time to articulate and agree upon this important assumption.

To summarize, this excerpt demonstrates two fundamental characteristics: the group is *validating* their model when they compare which gravitational force/constant they should be using, and the group is *assuming* things about their model when they say that the net force

Characteristic	Qualities
Conceptualizing	There needs to be some concept that a group is focusing on. Concepts usually range from individual physical quantities to more complicated physical relationships.
Representing	A particular concept should be represented mathematically. This process of representation usually involves translating a mathematical equation from the notes into a more general computer function.

Table 5.5: The characteristics and associated qualities pertaining to the computational practice of creating computational abstractions: representing physical concepts.

is comprised of only a gravitational force. Given these two characteristics, we feel confident in categorizing this excerpt as a strong illustration of the computational practice of assessing a computational model.

5.2.5 Creating abstractions

The computational practice of creating abstractions, as defined by Weintrop et. al, requires "the ability to conceptualize and then represent an idea or a process in more general terms." This ability show up frequently in the STEM domains – especially within introductory computational physics. The two fundamental characteristics of this computational practice, as summarized in Tab. 5.5, are: i) conceptualizing an idea and ii) representing it in more general terms. These two characteristics, if confidently observed within an excerpt, would serve to classify that excerpt as the computational practice of creating computational abstractions.

Consider Excerpt 13 from Group D in the following analysis. Overall, the group can be seen giving their net force a direction through the use of a unit vector ($\hat{r}$). They first recognize that their force needs to be a vector, and propose an equation to use that specifically involves a direction ($\vec{F} \propto \hat{r}/r^2$). Once they have their equation to work with, they begin to discuss how they can define it as a general function. In other words, the group can be seen *conceptualizing* and *representing* an idea in general terms.

They start by looking for an equation that they can use to try to calculate the net force on the satellite:

SA: *(108)* [calculating the magnitude of the force on his calculator]

SC: *(109)* Yeah just try that one equation first.

SC: *(110)* If that's not gonna work, then {I} think {the} other...

SD: *(111)* But the direction of F is {a vector}...

SD: *(112)* So we need to turn the r into a vector.

SC: *(113)* I think we should...

SD: *(114)* [writes force equation with $\hat{r}$]

Here the group can be seen deciding (or at least suggesting in line 109) that the computational force model that they are using will need to take a direction into account (i.e., it needs to be a vector). This equation, $\vec{F} \propto \hat{r}/r^2$ (retrieved from their notes), is written down on the WB. Notice that it involves using $\hat{r}$ to give the force a direction. This unit vector is the computational abstraction that the group identifies and ultimately begins to construct in their program. This abstraction helps them to work toward their goal of constructing the non-constant Newtonian gravitational force on the satellite.

Once the unit vector ($\hat{r}$) has been identified as a computational abstraction, they begin its creation in code:

SD: *(115)* So just put the r value, vector value...

SD: *(116)* Just put this [points to r hat] uhh function...

SB: *(117)* As a parameter?

SD: *(118)* Just give the computer a function so we don't have to calculate F like SA is doing.

SB: *(119)* That's a good idea.

Although they are clearly focusing on the concept of the direction of the Newtonian gravitational force, they are a little stuck on how to actually go about creating it. However, they at least know that they want it to be a function (line 118) rather than just a constant numerical value. Presumably, this is because they know that the numerical values will need to change in time (line 271):

SD: *(269)* No I mean this is the distance... and it has a direction...

SB: *(270)* So it's a vector.

SD: *(271)* Yeah this the position of the satellite is a vector.

SD: *(272)* Change with time...

SC: *(273)* Yeah I'm talk about the very beginning with the D... here [points to WB].

SB: *(274)* So the D is the radius...

To summarize this excerpt, the computational abstraction that the group has created is a function for the unit vector of the position of the satellite (line 116). They decide to create a function (as opposed to a hard-coded value) so that it will be able to change over time (line 272). That is, the group has *conceptualized* the direction of the force with a unit vector ($\vec{F} \propto \hat{r}$) and have *represented* that idea as position dependent and therefore more

Characteristic	Qualities
Isolating	The cause of an unexpected error that arises in a program must be tracked down. This sometimes involves retracing steps (or keystrokes) through the undo command, but usually involves testing the program through a process of guessing and checking.
Correcting	The unexpected error must ultimately be corrected in a long-term and generalizable manner.
Systematizing	When isolating or correcting the unexpected error, it should be done is a systematic and efficient way. This characteristic is not necessarily required.

Table 5.6: The characteristics and associated qualities pertaining to the computational practice of troubleshooting and debugging: isolating an unexpected error and correcting it in a systematic manner.

generalizable function (`rhat = satellite.pos/R`). Given these characteristics, this excerpt illustrates the computational practice of creating computational abstractions.

5.2.6 Troubleshooting and debugging

The computational practice of troubleshooting and debugging, as broadly defined by Weintrop et. al, refers to "the process of figuring out why something is not working or behaving as expected." This process is frequently undertaken by students in all fields of study – especially within introductory computational physics, given their reliance on incomplete/approximate computational and physical models. The three fundamental characteristics of this computational practice that we have identified, as summarized in Tab. 5.6, are: i) isolating an unexpected error, ii) correcting that unexpected error, and iii) doing so in a systematic/efficient way. These three characteristics, if confidently observed within an excerpt, would serve to classify that excerpt as troubleshooting and debugging.

For example, consider Excerpt 2 from Group I in the following analysis. Broadly, the group can be seen working to incorporate realistic values and generalizeable functions into their MWP. A couple of minutes into starting the problem (Sec. 3.4), they modify the pre-

written numerical value for the mass of the satellite from `1` to `1E4`. This leads, over the course of about thirty minutes, to the group defining the momentum of the satellite as a function. That is, the group can be seen *isolating* the cause of an unrealistic satellite trajectory and ultimately *correcting* it in a *systematic way* by redefining the momentum of the satellite from a hard-coded value to computer function.

The group begins by reading through the Euler-Cromer update of the position of the satellite in the calculation loop (line 6). This update involves the position of the satellite, the momentum of the satellite, the mass of the satellite, and the discrete time step (i.e., `satellite.pos = satellite.pos + satellite.p/msatellite*dt`):

SC: *(6)* It {the MWP} does the satellites position plus, vector, zero, five thousand, zero, thats the momentum of the satellite...

SC: *(7)* Divided by the mass, so, satellites position...

They also begin to consider the numerical values that have been assigned to the physical quantities being used (i.e., the initial position and momentum of the satellite and the mass of the satellite). Notably, the group points out (line 8) that the mass of the satellite should be changed to reflect the realistic value given in the problem statement:

SD: *(8)* This [points to the screen] is the mass? should we change that then?

SC: *(9)* Yeah we know that this is... they gave it to us didn't they?

SD: *(10)* Fifteen times ten to the third [reading from the problem statement].

SA: *(11)* I have all of the numbers up here [points to 4Q].

SC: *(12)* [changes the mass of the satellite from 1 to 1.5E4]

By changing the mass of the satellite from 1 to 1.5E4 (line 12), they have correctly modified the program to reflect the realistic situation presented to them. However, by changing the mass of the satellite they have also introduced an unexpected error – their satellite looks as if it is floating motionless in space.

After making their change to the program (line 12), the group begins to wonder (line 15) what the new visualization will look like. After some back and forth about what the visualization used to look like (line 18), they decide to run the program and observe the new visualization. The group discovers (line 20) that the satellite, although it used to travel in a straight line trajectory, now remains stationary relative to the rotating Earth:

SA: *(15)* Well I wonder what it {the visualization} looks like now...

SD: *(16)* It just like shoots straight.

SA: *(17)* Are you {sure}, did you already try it?

SC: *(18)* Yeah {previously}, but it might be different...

SD: *(19)* We just changed the mass.

SC: *(12)* [runs the program]

SA: *(20)* Uhh its not moving, maybe we should...

Given this unexpected error, the group begins to isolate the cause of the unexpected error. They consider that they may have introduced a syntax error since they last ran the program (e.g., in using E as opposed to **), resulting in it crashing the program (line 22). They also consider that changing the mass might have lead to the unexpected error, and work to at least temporarily rectify it (line 25):

SC: *(21)* We probably wrote it wrong...

SC: *(22)* Maybe it might have crashed the...

SA: *(23)* Well just exit out then.

SD: *(24)* Yeah.

SD: *(25)* Should we change it back and see if it runs again?

SC: *(26)* Well if we change it back to one it'll probably run again because we didn't change anything else.

SA: *(27)* Well can I see what it looks like when it runs with one?

SC: *(28)* Yeah.

Changing the mass of the satellite back to its initial dummy value is indeed a temporary fix to their unexpected error. However, a more long-term correction is needed to ensure the generalizability of their program. Ultimately, the group does work to correct the error in a more systematic and long-term manner:

SB: *(745)* So, okay so, we're all in understanding of why we are doing it like this {defining the momentum of the satellite as the mass times velocity} instead of declaring this {a hard-coded numerical value}?

SB: *(746)* It also like it makes it really explicit too, like when we go down here and do this thing where you take p divided by m you are literally just left with velocity...

SB: *(747)* So that's good.

SD: *(748)* Yeah.

Here, the group recognizes that the momentum of the satellite should be defined as a function utilizing the velocity and mass of the satellite separately (line 745). That way, when the momentum is used in the Euler-Cromer update, it will correctly divide out the mass no matter what value they use (line 746).

The type of systematic correction of an unexpected error seen in this excerpt can be contrasted with our motivating case study (Sec. 4). That is, the changes that the group made in the case study could be characterized as a more haphazard approach, as opposed to the present excerpt where the group shows a certain level of reasoning behind their actions (line 746). Accordingly, this excerpt seems to illustrate a group working in a systematic/efficient way as they troubleshoot and debug their program.

To summarize, the unexpected error that the group runs into is that in changing the mass of the satellite to reflect the realistic situation, the satellite remains motionless relative to the rotating Earth (line 20). This introduces concern to the group, presumably because a straight line trajectory is closer to a geostationary orbit as compared to no trajectory at all. The group works to *isolate* the error by changing the mass of the satellite back to its initial dummy value and finding that this does indeed rectify the unexpected error (line 25). Ultimately, the group works to *correct* this error first temporarily by changing the mass of the satellite, and then more *systematically* and permanently by redefining the momentum of the satellite as a function (line 745). Given these characteristics, this excerpt illustrates the computational practice and process of troubleshooting and debugging.

5.2.7   Thinking in levels

The computational practice of thinking in levels, as defined by Weintrop et. al, involves the analysis of a system that ranges "from a micro-level view that considers the smallest elements

Characteristic	Qualities
Leveling	A group should either implicitly or explicitly define the different levels of a system. For example, every MWP can be broken down into an initial condition level and a calculation loop level.
Featuring	The unique features of each level should be articulated by the group. For example, a group might articulate that physical quantities that need to change in time must be placed in the calculation loop.

Table 5.7: The characteristics and associated qualities pertaining to the computational practice of thinking in levels: breaking a program into different levels and attributing features to them.

of the system to a macro-level view that considers the system as a whole." This type of high- and low-resolution analysis of a system is a skill that shows up frequently in scientific disciplines – and especially within the domain of computer science. The various control structures common to computer programming (e.g., a while or a for loop) must not only work independently (i.e., at the micro-level) but must also work together (i.e., at the macro-level) with other control structures to produce the desired results of the program. Accordingly, the two fundamental characteristics that we have identified for this computational practice, as summarized in Tab. 5.7, are: i) identifying the different levels of a system and ii) correctly attributing features of that system to the appropriate level.

For example, consider Excerpt 6 from Group A in the following analysis. Broadly speaking, this excerpt focuses on the group making decisions about what needs to be added to their code and, more importantly, where those things needs to be added. More specifically, they work to construct a function for the momentum of the satellite (which depends on its velocity) as well the net force acting on it.

Early on, the group decides that they should construct a function for the momentum of the satellite in their program (line 76):

SC: *(74)* umm so we have like its defining p of the satellite, and thats like p is momentum

you know? Like p equals m v

SC: *(75)* but theres nothing in here that actually defines the p of the satellite as being m v

SC: *(76)* so I feel like we need to put in a v, and then the velocity of the satellite is a variable

SC: *(77)* and then make the momentum of the satellite as a combination of the mass and velocity

SB: *(78)* umm my question for you, from the perspective of...

SB: *(79)* were doing circular motion, and as you go around from point a to b, your velocity is changing cause its changing direction

SB: *(80)* maybe, I guess we can define speed, but uhh the trick with velocity since its going to be changing

SB: *(81)* like you want the variable to continue changing

SB: *(82)* and for the variable to continue updating you have to put it in the calculation loop

SC: *(83)* umm okay

However, this raises the issue of where to actually place the function in the code (line 78). The group decides that they must define the velocity inside the calculation loop (line 82) given that it must "continue updating" as its direction continues to change. The crucial feature that the group is articulating here is that the calculation loop is where time-dependent or changing quantities must be placed.

After a short TA interaction focusing on the generalizability of their program, the group returns to topic of where certain things are/should be placed in their code:

SB: *(107)* may I umm, may I uhh...

SB: *(108)* okay so, there is like, theres two sections in the code...

SB: *(109)* so in the code, you have your calculation loop and your parameters and initial conditions.

SB: *(110)* so from what we have, were defining our initial conditions as this model right here, which is just Earth and the satellite like its defined these two bodies and it has set the momentum of this

SB: *(111)* and then I was thinking, in the code here in the calculation loop the force is set to zero zero zero, so were never defining F net at any point.

SB: *(112)* I think what we need to do is describe F net. The only other thing we have to delcare is the radius

SC: *(113)* yeah sure we could do that

SD: *(114)* okay

Here, the group clearly articulates (line 108) the two different sections/levels of the program (i.e., the initial conditions and the calculation loop) and details some of the components belonging to each level. That is, the objects of the Earth and the satellite belong to the initial conditions (line 110) and the net force acting on the satellite belongs to the calculation loop (line 111).

91

To summarize, the group has broken their program into the two different *levels* of initial conditions and calculation loop (line 108). Similarly, they have attributed the particular *feature* of time-dependence to the calculation loop (line 82). Given these two characteristics, this excerpt is can be used to illustrate the computational practice of thinking in levels.

5.2.8 Communicating information

The computational practice of communicating information, according to Weintrop et. al, usually involves a visualization or representation (e.g., a graph) that can be used to "highlight the most important aspects of what has been learned about the system in such a way that it can be understood by someone who does not know all the underlying details." This communication skill is especially important in fields involving complex and interrelated systems, such as those observed in physics and engineering. The ability to share useful information with colleagues without going through all of the underlying details and mechanisms is crucial. Accordingly, the two fundamental characteristics associated with this particular practice, as summarized in Tab. 5.8, are: i) a general process of communication (detailed in Tab. 5.8) and ii) the demonstration of an understanding that has been reached about the system.

For example, consider Excerpt 30 from Group E. At this stage, the group has begun to construct a Newtonian gravitational force model, but is struggling with its implementation. A brief interaction with the TA shows them communicating information about their understanding of the underlying concept of circular motion, as well as an understanding of the power and generalizability of the program. After this interaction, the group continues with the construction of the Newtonian gravitational force, and more specifically, its direction.

About halfway into the program, the group is struggling (line 231) to construct their

Characteristic	Qualities
Communicating	The act of communication can range from pure dialogue between two or more individuals to detailed visualizations that capture the relevant information to be shared. For example, creating a graph of a physical quantity vs. time can be used to succinctly share information about the time dependence of that physical quantity. Alternatively, this time dependence could be articulated verbally through dialogue.
Understanding	The information being communicated should demonstrate an understanding that the group has of the underlying mechanics. For example, a group might communicate the way that the position, force, and momentum of the satellite are interrelated as simulated time progresses.

Table 5.8: The characteristics and associated qualities pertaining to the computational practice of communicating information: a general process of communication that demonstrates an understanding.

Newtonian force model. The TA recognizes that they need a little help, and asks for them to explain their process (line 232):

SB: *(231)* Physics man... this is a mess [points to scratchwork on WB].

TA: *(232)* No no, go ahead and explain...

SB: *(233)* Okay, so...

SB: *(234)* With that beautiful little formula right here [points to Newtonian force equation]...

SB: *(235)* We decided... this force has to be negative.

SB: *(236)* Because our initial momentum is five thousand in the positive y-direction.

TA: *(237)* Okay, I can dig that.

SB: *(238)* And then our unit vector {for position} right now, is one zero zero [inaudible].

SB: *(239)* So if we have the force multiplied by that, negative, so it has to be negative.

SB: *(240)* Then this {the momentum/velocity} will slowly start approaching negative five thousand here in the x-direction.

SB: *(241)* And then once that reaches {negative} five thousand, then our position is here at zero one zero...

SB: *(242)* And then since it's {the force} negative, it'll move it downward.

SB: *(243)* And that will happen at every step [draws four points on a unit circle].

TA: *(244)* Okay good.

Thus, a member of the group can be seen communicating information (lines 234-243) about the way that the force, momentum, and position are related at various points on the x- and y-axes for a circular trajectory. Although just one member of the group is doing a majority of the communication, they are acting as a spokesperson or a representative for the group (line 235). This information shows a clear understanding of the rather complicated interrelation (i.e., sinusoidal and out-of-phase) of these physical quantities.

However, the TA continues to press them on their understanding (lines 245, 248, and 250) by asking them to consider positions other than those on the x- and y-axes:

TA: *(245)* What about here [draws a dot in the first quadrant]?

SD: *(246)* Point five...

SB: *(247)* Then it'll be... the square root of two, square root of two, zero.

TA: *(248)* Okay, what about here [draws a dot in the second quadrant]?

SB: *(249)* Square root of two, negative square root of two, zero.

94

TA: *(250)* What if it's not at forty five degrees?

TA: *(251)* What if it's just at some arbitrary angle?

SB: *(252)* Well, the reason that were doing this...

SD: *(253)* The r hat is gonna update as it goes.

TA: *(254)* Okay...

SD: *(255)* So you don't need to know that.

SB: *(256)* Yeah you don't need to know that...

TA: *(257)* Okay, that's fine.

In response, the group demonstrates a strong understanding of not just the interrelation between physical quantities (lines 234-243), but also of the computational power of their program. That is, another member of the group articulates (line 253) that their definition of the direction of the force in code (i.e., $\hat{r}$) will automatically update to account for these various/arbitrary positions.

To summarize, this excerpt shows a TA interaction focusing on the construction of the Newtonian gravitational force acting on the satellite. The group can be seen *communicating* information about the interrelation of the position, force, and momentum of the satellite. Through this dialogue, they are demonstrating a clear *understanding* of these interrelations, as well as a clear understanding of the power and generalizabilty of their program. Given this communication and demonstration of understanding, this excerpt can be classified as an illustration of the computational practice of communicating information.

Chapter 6

Discussion

This chapter provides a discussion of the overall findings within our data, the limitations of those findings and the underlying framework, as well as some future research and instructional opportunities that are made available.

6.1   Findings

In the sections that follow, we present our findings of the common, less common, and unobserved practices within our data set. The common practices are those that were identified at least three times in a majority of the groups (individual practices occurred between zero and seven times per group, with an average of three occurrences). The less common practices are the remaining of the observed practices. The unobserved practices were not identified at all. Along with some of the statistics of the frequencies of these practices (i.e., raw numbers and percentages), we provide their definitions and reference detailed examples. Additionally, and perhaps most importantly, we discuss the reasons why a particular practice might have a given frequency.

6.1.1 Common practices

Eleven of the practices laid out by the framework have been identified multiple times in eight of the groups that were analyzed. Accordingly, these practices (listed in Tab. 6.1) are deemed common and are discussed below. It is important to pay attention to these practices because, as instructors, we not only want students to accomplish tasks, but we also want to make sure that they are engaging in things like critical and computational thinking. Being able to identify and encourage these practices as they occur (or don't occur) in a classroom, therefore, is crucial to effective pedagogy and course design. Accordingly, we must develop clear and reliable definitions for each of the common practices.

Category	Practice	Number	% of category	% of all
Data	Creating data	27	31	9
	Analyzing data	23	27	8
	Visualizing data	36	42	13
Modeling	Designing models	32	33	11
	Constructing models	18	19	6
	Assessing models	27	28	9
Problem solving	Creating abstractions	22	27	8
	Programming	21	26	7
	Troubleshooting and debugging	24	29	8
Systems	Thinking in levels	18	33	6
	Communicating information	23	43	8

Table 6.1: The computational practices that have been deemed common are shown with the number of times each practice was identified, the percentage of its category that it occupies (i.e., the number of times a practice was observed divided by the total number of practices from that category), and the percentage of all the practices that it occupies (i.e., the number of times a practice was observed divided by the total number of practices from all categories). Horizontal dividers separate the different categories (i.e., data, modeling, problem solving, and systems thinking).

The practice of **creating data** involves the construction of an *automatic* or algorithmic process that will quickly produce a large set of data and using that set of data to *advance* toward their goals. For example, constructing an Euler-Cromer algorithm to create a set of

data representing the position of the satellite over time advances the group toward their goal of simulating the orbit of the satellite (see Sec. 5.2.1). This type of practice was observed 27 times across the data set, accounting for 31% of the data practices, and 9% of all practices.

Creating data is expected to show up commonly in our data given the learning goal of using mathematical and computational thinking (P4). We wanted students to take advantage of the Euler-Cromer algorithms to generate the sets of data representing the position and momentum of the satellite over time. We also wanted them to construct and use different models to generate the set of data representing the force over time. These algorithms and models require a lot of mathematical and computational thinking, aligning well with that learning goal.

Additionally, the problem cannot be solved analytically with introductory level mathematics. However, it can be solved numerically with introductory level mathematics and computation. For example, consider Excerpt 7 from Group C where the TA is prompting the group to create data:

TA: : But you need the force to keep changing direction as it moves around

SC: : Right

TA: : So you can't just hard core the numerical value that you found last time

SC: : Oh... because this position of the satellite is going to change, which means the force is going to change...

TA: : Exactly

SB: : Oh, gotcha

Here, the tutor facilitating the creation of data by focusing on the way that the force needs to continually change as the satellite moves. These types of interactions usually initiate the process of designing, constructing, and assessing computational models and algorithms that ultimately create large sets of data.

Overall, we want students in P^3 to be able to use simple control structures with force models of varying complexity to generate large sets of data for complicated and realistic motion problems.

The practice of **analyzing data** involves a broad process of analysis that includes sorting data into *categories*, looking for *trends*, looking for *correlations*, and/or identifying *outliers* that can be used to reach some *conclusion*. For example, when a print statement is used to verify that the force acting on the satellite has the trend of remaining constant in simulated time, a conclusion can be drawn about the correctness of the underlying force model (see Sec. 5.2.2). This type of practice was observed 23 times across the data set, accounting for 27% of the data practices, and 8% of all practices.

Analyzing data is expected to show up commonly in our data given the learning goal of analyzing and interpreting data (P3). We recognize that large sets of data need to be generated using computational algorithms and models, and that these sets need to be analyzed in order to asses and validate the underlying algorithms and models. There are many different ways to analyze data, but it usually leads to some interpretation or conclusion that is made. Given the utility of analyzing data when it comes to designing, assessing, and constructing the underlying computational models, we expect to see this practice commonly in our data.

One technique of analysis that is often suggested is to use a print statement in the calculation loop so the data itself can be analyzed. For example, consider Excerpt 19 from Group G where the TA makes this type of suggestion:

TA: : check like, so i know you know how to do this... use a print statement, check if its doing anything, make sense of where its not, or if its running or if its not running

SB: : Yeah, okay

TA: : Talk everybody through what you're doing though...

SB: : Yeah, I will.

Here, the TA suggests that they use a print statement so that they can analyze the data representing the force acting on the satellite and to make decisions based on that analysis. These types of TA interactions usually initiate the practice of analysis of a set of data.

Overall, we want students in P^3 to be able to interpret and attach meaning to the patterns that can be found in large sets of data.

The practice of **visualizing data** involves the *production* of a visualization that clearly *conveys* some information. For example, the computational production of a dynamically updating graph of the distance between the satellite and the Earth vs. simulated time can be produced and used to clearly convey information about the nature of the orbit (i.e., how close the orbit is to perfectly circular). This type of practice was observed 36 times across the data set, accounting for 42% of the data practices, and 13% of all practices.

Visualizing data is expected to show up commonly in our data given the learning goal of analyzing and interpreting data (P3). One of the ways that data can be analyzed and interpreted is with a visualization. Specifically, the visualizations we see students making are that of the trajectory, the force, the momentum, and the graph of distance vs. time. These visualizations efficiently convey information both to the students and to the TA (e.g., the visualization of the force conveys information about its central nature).

100

Additionally, students have been working with MWPs to produce dynamic visualizations of motion since the first week of class. The first and second computational problems, focusing on boats and hovercrafts, respectively, were visualized in a number of ways (e.g., producing visualizations of their trajectories). In other words, students are familiar with the visualization of data coming into the third problem. Not to mention, the problem statement, shown in Fig. 3.4, explicitly asks students to produce a simulation/visualization of an elliptical orbit.

Furthermore, after a group has correctly constructed the Newtonian gravitational force, many of the tutor interactions focus on the generation of a graph to clearly show that the satellite isn't traveling in a perfectly circular orbit. For example, consider Excerpt 38 from Group A where the TA is prompting the group to add in a graph to their code:

TA: : I'd like you to graph the orbital... the magnitude of the radius of the orbit vs. a function of time...

SC: : Okay...

TA: : And have that graphed as well and it updates

SC: : Okay.

The tutor is presenting the additional goal of producing a graph to the group. This graph can be used to efficiently convey information about how close the satellite is to perfectly geostationary.

Among all things, we want P^3 students to understand that computers can be used to quickly generate visualizations that can easily be tweaked, and that those visualizations can be useful when it comes to understanding and communicating the physics of the realistic

phenomenon being modeled. Accordingly, we are likely to observe the visualization of data in our data.

The practice of **designing computational models** involves *defining* the individual components of a model, *relating* the model to the physical phenomenon under investigation, and articulating what *predictions* the model will be able to make. For example, the mass of the satellite, the magnitude of its velocity, the radius of its orbit, and the polar angle that it makes can all be separately defined in code. Additionally, these individual components can be combined, following an equation, to produce the expected oscillatory motion of the satellite. Finally, the resulting force model can be used to make predictions about the motion of the satellite (see Sec. 5.2.3). This type of practice was observed 32 times across the data set, accounting for 11% of the data practices, and 33% of all practices.

We expect to see this practice commonly in our data given the learning goal of developing and using models (P1). The course was specifically designed to focus on different force models with a range of complexities. That is, the course focuses on a constant zero force, a constant non-zero force, and a non-constant force model. Given that the students must actually develop these models in code, we frequently observe them designing computational models.

Further, the four-quadrants are meant to scaffold the design process by highlighting the knowns, unknowns, and assumptions of the model. This scaffolding often facilitates the design process by helping groups to define the individual elements of their model. For examples, consider Excerpt X from Group Y where one student is clear to articulate the individual elements they are defining by writing them on the four-quadrants:

SA: So I'm just gonna go ahead and define those over there then

SA: [writing on 4Q]

SA: Should we do that?

SB: Yeah go ahead and... we have the mass

SB: and the position of the satellite

SC: and the velocity from last time

SA: Okay hold on [writing them down]

Here, the individual elements of the mass, position, and velocity of the satellite are individually defined. That is, we see students using the four-quadrants to help them design their model.

Overall, we want students in P^3 to be able to define the individual elements of a model, relate them to each other, and make predictions with computational force models.

The practice of **constructing computational models** involves *implementing* new behavior in code by either *creating* a new model or by *extending* a previously written model. For example, implementing an attraction between two massive objects in code can be achieved through the construction of a force model. This behavior can be implemented in one shot (e.g., immediately constructing a Newtonian gravitational force that can handle elliptical orbits) or can be implemented by successively extending an approximate model (e.g., moving from a constant gravitational force that generates a parabolic trajectory, to a centripetal force that generates a circular orbit, to a Newtonian gravitational force that generates an elliptical orbit). This type of practice was observed 18 times across the data set, accounting for 19% of the data practices, and 6% of all practices.

Constructing computational models is expected to show up frequently within our data given the learning goal of developing and using models (P1). Developing a model in code

invariably requires students to map mathematical equations onto VPython syntax. This involves using proper operations (e.g., adding, multiplying, calculating magnitudes), using proper order of operations (e.g., using parentheses to clear up any ambiguity), and ensuring computational abstractions are of the proper type (e.g., that position is a vector, or that distance is a scalar). Given that these things must all be constructed in code, we frequently observe students constructing models.

Additionally, many tutor interactions are intended to facilitate this practice. For example, consider Excerpt 9 from Group I where the group has designed their model and is beginning to construct it in code:

TA: No, what you have their on the whiteboard looks good...

SD: Okay so we just need to like take this equation and like

SD: put it in the program

TA: Right...

SD: Right, but how do we do that...

SC: So just take big G... and then like multiplied times...

SC: m sat, err, yeah the mass of the satellite

SA: Okay... [begins to type]

Here, the model they have designed is the Newtonian gravitational force and they begin to construct it in code in terms of the universal gravitational constant, the mass of the satellite and the Earth, and the satellite's position relative to the Earth. Given these types of interactions, we frequently observe students constructing models.

Overall, we want students in P^3 to be able to construct models in code, whether or not the models are correct.

The practice of **assessing computational models** involves identifying the *assumptions* built into a model and *validating* them by comparing to reality to ensure predictive accuracy. For example, groups frequently assume that the orbit of the satellite will be perfect circular. Although this assumption is a good starting point, it is invariably checked for validity when considering arbitrary initial conditions that lead to more general elliptical orbits (see Sec. 5.2.4). This type of practice was observed 27 times across the data set, accounting for 28% of the data practices, and 9% of all practices.

Assessing computational models is expected to show up frequently within our data given given the learning of developing and using models (P1). Once a model has been designed and constructed to a reasonable degree, it can be used to generate information (e.g., a trajectory of the satellite). This information can ultimately be used as evidence to make an argument for or against the validity of that model. Thus, throughout the process of designing, constructing, and most importantly assessing a computational model, students should be engaging in argument based on evidence.

Many tutor interactions can help to facilitate this practice. For example, consider Excerpt X from Group Y where they articulate an assumption built into a model and validate its use given prompting:

TA: Yeah but when is that equation good?

SB: When its in free...

SC: Like when its falling

TA: Right close to the Earth

SB: Yeah

SC: Which is why we have that written here under assumptions {on the 4Q}

TA: Okay good but... is that what you have over here?

SB: No

SC: No we need a different equation...

Here, the poor assumption is that of a uniform gravitational acceleration, which invalidates their model. Given these types of tutor interactions, we expect to frequently observe students assessing models.

Overall, we want students in P^3 to be able to validate different computational models by identifying their assumptions, whether or not they did the design and/or construction themselves.

The practice of **creating computational abstractions** involves taking a physical *concept* and *representing* that concept in code. For example, the physical concept of the unit vector giving a proper direction to the Newtonian gravitational force acting on the satellite can be most easily represented in code by combining the position of the satellite and its magnitude (see Sec. 5.2.5). This type of practice was observed 22 times across the data set, accounting for 27% of the data practices, and 8% of all practices.

Creating computational abstractions is expected to show up frequently within our data given the learning goal of being able to develop and use models (P1). All of the models used in the course (i.e., the various force and motion models) have some mathematical form that can be translated into VPython syntax. That is, in order to construct a computational model, you must first create the computational abstractions that it depends on. Given the focus on

modeling in the course, we expect to commonly observes students creating abstractions.

Additionally, some of the tutor interactions can facilitate this practice. For example, consider Excerpt 9 from Group F where the tutor questions them on the definitions that they have in their code:

TA: So I see that you have those things defined on your whiteboard

TA: But where do you have those defined in the code?

SA: But thats what I'm saying, thats what were working on

TA: Okay, so what are you thinking then?

SA: We have these things [points to board] defined...

SA: And were gonna like input those values for those variables

Here, the definitions that they have on the whiteboard are the mass of the satellite, its speed, and radius of circular orbit. This interaction ultimately prompts them to make corresponding computational abstractions in code. Given these types of interactions, we expect to commonly observe this practice in our data.

Ultimately, we want students in P^3 to be able to make abstractions in code when dealing with various physical concepts.

The practice of **computer programming** involves *modifying* code while *arranging* that code in proper syntax. For example, while modifying the force model in the calculation loop, all lines must be arranged with the proper indentation. In other words, aside from the validity of the force model, the syntax must be in order for the computer to be able to interpret things correctly and to run without error. This type of practice was observed 21 times across the data set, accounting for 26% of the data practices, and 7% of all practices.

107

Computer programming is expected to be commonly observed in our data given the learning goal of using mathematical and computational thinking (P4). Groups are working with MWPs in VPython (see Sec. 3.4.2.1), which comes with its own unique syntax that must be adhered to strictly. Although the syntax in VPython is very intuitive (e.g., calculating the magnitude of a vector can be done by calling the `mag()` function), small and sometimes difficult to find syntax errors (e.g., a missing parenthesis) can lead to frustrating runtime errors. Given these difficulties, we expect to see students engaging frequently in this practice.

Additionally, this practice is heavily scaffolded through tutor interactions. Given that many students have little to no prior programming experience, tutors sometimes guide students in their programming. For example, consider Excerpt 12 from Group B where the group knows what to do, but is unsure of how to program it:

SB: : TA, we need help

TA: : Okay I can try

SB: We don't know how to like take the magnitude of this

TA: where...

SB: right here, in our force, equation for the force

TA: ahh okay you need to put parentheses.

SB: where here?

Here, the tutor is reminding the group that the proper syntax that must be adhered to requires parentheses. Given these types of interactions, we frequently observe groups to be engaging in the practice of computer programming.

108

Overall, we want students in P^3 to have experience with programming and the difficulties associated with it.

The practice of **troubleshooting and debugging** involves *isolating* an unexpected error in the code, *correcting* that error in a long-term and generalizable manner, and doing so in a *systematic* fashion where applicable. For example, without defining the initial momentum of the satellite as a function in terms of its previously defined mass and initial velocity, changing the mass of the satellite won't correctly propagate through the program, leading to unexpected and undesirable results. Systematically isolating the cause of this error (i.e., not defining the momentum in a dynamic way) allows for it to not only be corrected, but to be corrected in a long-term and generalizable manner (see Sec. 5.2.6). This type of practice was observed 24 times across the data set, accounting for 29% of the data practices, and 8% of all practices.

Troubleshooting and debugging is expected to show up frequently within our data given the learning goal of being able to develop and use models (P1). During the process of developing and using a model, unexpected errors frequently occur and must be corrected. These unexpected errors can involve things like syntax errors or unexpected/unphysical behavior. In either case, students must identify those errors, and ultimately correct them in a systematic manner. Given this focus on developing and using models, we expect to see this practice commonly in our data.

Further, many tutors intentionally guide groups as they troubleshoot and debug. For example, consider Excerpt 22 from Group H where the tutor points out that their force model in code does not match their force model on the board:

TA: Oh I see what it is

SB: What

TA: Okay so in the denominator of your force, you have the magnitude of the position of the satellite

SB: Right

TA: But what do you have on your board

SB: Ohhhh

SC: We need it squared

Here, the incorrect force model produces an extremely large force that rapidly accelerates the satellite to ludicrous speed. This small error, although syntactically correct, produces unphysical results. Given these types of tutor interactions, we expect to see this practice commonly in our data.

Overall, we want students in P^3 to be able to handle unexpected errors that arise while programming, whether they be syntactical or physical.

The practice of **thinking in levels** involves breaking the MWP into different *levels* and attributing those different levels with their characteristic *features*. For example, the program as a whole can broken down into the two different levels of the initial conditions and the calculation loop (see Sec. 5.2.7). Each level has its own defining features: the initial conditions level is where time-independent computational abstractions can be defined, whereas the calculation loop is where time-dependent computational abstractions must be defined. This type of practice was observed 18 times across the data accounting for 33% of the systems thinking practices, and 6% of all practices.

Thinking in levels is expected to show up frequently within our data given the learning goal of developing and using models (P1). The Newtonian gravitational force model and Euler-Cromer motion algorithms constitute a model of motion that must be developed in code and ultimately used for some purpose. While students are developing this model of motion, they must maintain the overall structure of the MWP written in VPython (see Fig. 3.4.2.1) – without proper structure and syntax, the program as a whole runs into fatal errors. This structure, that must be maintained, is naturally broken down into several different levels: the objects, initial conditions, time set-up, and calculation loop. These levels are indicated in the MWP with comments (e.g., `#Calculation Loop`), and each level has its own unique features. Maintaining these features for each level is critical to a runnable program.

Additionally, students are introduced to the concept of iterative prediction of motion as an algorithmic change in different physical quantities over time. Specifically, $\vec{p}_{\text{new}} = \vec{p}_{\text{old}} + \vec{F}_{\text{net}}\, dt$, $\vec{r}_{\text{new}} = \vec{r}_{\text{old}} + \vec{v}\, dt$, and $t = t + dt$, as described in the course notes (see Sec. 3.1). These time-dependent physical quantities can be contrasted with time-independent (or approximately time-independent) physical quantities (e.g., the local acceleration due to gravity). Identifying the correct time-dependence of a physical quantities is necessary to ensuring proper placement of its definition – time-independent quantities can be placed in the initial conditions level, whereas time-dependent quantities must be placed in the calculation loop. For example, consider Excerpt 12 from Group C:

SA: do we need F net to be calculated inside the loop?

SA: that is do we need to recalculate F net every time? is it changing?

SB: no

SA: so we could just throw it outside of the loop

Here, the students (incorrectly) articulate that the net force does not need to placed in the calculation loop because it does not need to update. That is, they identify the different levels of inside and outside the loop, and correctly attributed the feature that updating quantities must be placed inside the loop, whereas other can be placed outside.

Above all, we want students in P^3 to understand the difference between time-dependent and time-independent physical quantities, and to be able to properly define and place them in code. Accordingly, we observe this practice commonly in our data.

The practice of **communicating information** involves the broad process of *communication* that ranges from pure *dialogue* to self-contained *visualizations* that communicate some *understanding* that the group has achieved. For example, an understanding of the complicated but powerful computational interrelation between the force, position, and momentum of the satellite is frequently communicated verbally within and beyond groups (see Sec. 5.2.8). This type of practice was observed 23 times across the data accounting for 43% of the systems thinking practices, and 8% of all practices.

Communicating information is expected to show up frequently within our data given the learning goal of being able to obtain, evaluate, and communicate information (P7). Once information has been obtained and evaluated, it is crucial to ensure that each member of the group can communicate an understanding of it. Accordingly, students are required to continually explain their thought process throughout the day. Given this focus on encouraging explanation, we expect to frequently observe students communicating information in our data.

Further, many tutor interactions can help to facilitate this practice. For example, consider Excerpt 30 from Group E where the TA continues presses them on their understanding of

the direction of the force by asking them to consider positions other than those on the x- and y-axes:

TA: *(245)* What about here [draws a dot in the first quadrant]?

SD: *(246)* Point five...

SB: *(247)* Then it'll be... the square root of two, square root of two, zero.

TA: *(248)* Okay, what about here [draws a dot in the second quadrant]?

SB: *(249)* Square root of two, negative square root of two, zero.

TA: *(250)* What if it's not at forty five degrees?

TA: *(251)* What if it's just at some arbitrary angle?

SB: *(252)* Well, the reason that were doing this...

SD: *(253)* The r hat is gonna update as it goes.

TA: *(254)* Okay...

SD: *(255)* So you don't need to know that.

SB: *(256)* Yeah you don't need to know that...

In response to the TA prompting, the group demonstrates a strong understanding that their definition of the direction of the force in code (i.e., $\hat{r}$) will automatically update to account for these various/arbitrary positions. Given these types of tutor interactions, we frequently observe this practice in our data.

Overall, we want students in P^3 to be able to clearly communicate their understanding of physical concepts, both through dialogue and visual representations.

113

6.1.2   Less common practices

Five of the practices laid out by the framework have been identified just a couple of times and in only five of the groups that we analyzed. Accordingly, these practices (listed in Tab. 6.2) are deemed less common and are given a reasonable amount of attention. Table 6.2 is broken down into the different categories: the data practices, modeling practices, problem solving practices, and systems thinking practices. the data practices were either unobserved or commonly observed, and so this table shows only the less commonly observed modeling, problem solving, and systems practices.

Category	Practice	Number	% of category	% of all
Modeling	Understanding concepts	7	7	2
	Finding and testing solutions	13	13	4
Problem Solving	Assessing solutions	9	11	3
Systems thinking	Investigating systems	13	13	4
	Understanding relationships	13	13	4

Table 6.2: The computational practices that have been deemed less common are shown with the number of times each practice was identified, the percentage of its category that it occupies, and the percentage of all the practices that it occupies. Note that none of the data practices were less commonly observed.

The practice of *understanding concepts* involves *progressing* toward a deeper understanding of a concept by *interacting* with a computational model. For example, while designing, constructing, or assessing a Newtonian force model in code, students progress in their understanding of the abstract concept of a unit vector as providing purely a direction to a physical quantity. This type of practice was observed 7 times across the data accounting for 7% of the systems thinking practices, and 2% of all practices.

We expect to see this practice in our data given the learning goal of engaging in argument from evidence (P6). Specifically, individuals must be able to defend their understanding of various physical concepts while using their program as evidence. Each program produces a

number of piece of evidence (e.g., graphs, numerical values, visualizations, etc.) that can be used to support claims of understanding. Accordingly, we expect to see students engaging in this practice.

Further, many tutor interactions help to facilitate this practice. For example, consider Excerpt 12 from Group C where the TA is asking them to illustrate their point with their program:

SA: The force has to point in the same direction as the momentum of the satellite...

TA: Yeah but why are you saying that?

TA: Can you use your program to sort of prove that to me?

SA: Yeah, so, if you look at the arrows on the satellite.

SA: They always like move toward the Earth.

SA: So we know that the force has to be pointing toward the Earth.

SA: So our direction has to be correct....

TA: Okay... okay that makes sense.

. Here, a student clearly uses the visualization of the momentum of the satellite to demonstrate her understanding of the relationship between the force and the change in momentum. Given these types of interactions, we expect to see groups understanding concepts in our data.

Overall, we want students in P^3 to be able to use computers to learn physical concepts, and to be able to communicate their understanding with them.

The practice of *finding and testing solutions* involves *justifying* the use of a particular solution. Often, the particular solutions that we see are the different force models covered

in the course notes (e.g., local gravitational force). As students progress from incorrect or approximate force models (i.e., the local gravitational or centripetal) to the correct model (i.e., Newtonian gravitational force), we see them continually testing along the way. For example, a group might recognize that the local gravitational force model does not allow for the satellite to travel in a bound orbit, and move on to searching for a new model. This type of practice was observed 13 times across the data accounting for 13% of the systems thinking practices, and 4% of all practices.

We expect to see this practice in our data given the learning goal of obtaining, evaluating, and communicating information (P7). Most importantly, groups are required to evaluate information by using their program to make predictions and to evaluate it in terms of its predictive validity. If the model is not justified, a new model must be sought out, and the process of testing begins again. Given this focus on evaluating information when it comes to the justification of a solution, we expect to see this practice in our data.

Additionally, many tutor interactions can facilitate this practice. For example, consider Excerpt 20 from Group A where they are using a local gravitational force model with a deceptive satellite trajectory:

TA: So the problem is...

TA: If you look at your force, you have a local gravitational force...

TA: But that is only good when?

SB: When its close to Earth

TA: And is that what we have here?

SA: But it looks like its orbiting [points to screen]

TA: Its actually parabolic, I know it looks like its gonna orbit but its not

SB: Oh cause the force here is only in the x direction

TA: Right... so you need to change that...

Here, the tutor is prompting them to justify their force model by scrutinizing the resulting trajectory. Given these types of interactions, we expect to see this practice in our data.

Ultimately, we want students in P^3 to be able to find different force model and to be able to justify them in terms of the applicability to various situations.

The practice of *assessing solutions* involves the *comparison* two or more different solutions. This is different than just testing solutions, given that it focuses on a comparison between two or more solutions. For example, groups often compare the expected behavior using a local gravitational force to using a Newtonian gravitational force. In this way, multiple solutions are assessed in terms of their validity. This type of practice was observed 9 times across the data accounting for 11% of the systems thinking practices, and 3% of all practices.

We expect to see this practice in our data given the learning goal of being able to construct explanations (P5). That is, we not only wanted students to make comparisons between models, but we also wanted them to clearly explain those differences. These long and often complicated explanations, then, are often a clear indication of a group assessing a solutions.

Additionally, many of the tutor interactions can encourage this practice. For example, consider Excerpt 19 from Group H where the TA is asking them about the two models they have written on their board:

TA: So these two forces that you have on your board...

117

TA: Fg and Fcent...

TA: Which of those do you want to use?

SB: We are thinking Fcent...

TA: Why?

SB: Because we have everything we need for it...

Here the models that the group is comparing are the gravitational force and the centripetal force. Specifically, the group explains the difference between the two as being in terms of the requirements of the model. Given interactions like these, we expect to see this practice in our data.

Overall, we want students in P^3 to be able to explain the differences, both qualitative and quantitative, between various force models.

The practice of **investigating systems as a whole** involves *questioning* and *interpreting* data gathered from a system as a whole. For example, a graph of the set of data representing the distance between the Earth and the satellite can be questioned about its qualitative time-dependence (e.g., if it is constant, linear, quadratic, sinusoidal, etc.). This type of practice was observed 13 times across the data, accounting for 13% of the systems thinking practices, and 4% of all practices.

Investigating systems as a whole is expected to show up in our data given the learning goal of planning and carrying out investigations (P2). The act of planning is scaffolded by the four-quadrants – students must list their knowns, unknowns, assumptions, and draw out any representations. These quadrants help students to generate questions (e.g., "what are we even trying to figure out?") that can be investigated for answers. And, given the focus

on questioning in Weintrop et. al's definition of investigating systems as a whole, we expect to see this practice in our data.

Additionally, investigating systems as a whole likely shows up in our data given the learning goal of analyzing and interpreting data (P3). Many sets of data need to be created (see Sec. 5.2.1) in the calculation loop. Ultimately, these sets of data need to be analyzed (e.g., visually through a graph or manually through a print statement). For example, consider Excerpt 49 from Group I, where a graph is used to generate meaning in their data:

TA: So if you see that wobble there [points to graph]

TA: And, and so what does that tell you about the orbit?

SB: That its not perfectly circular?

SC: Right that it doesn't go in a perfect circle.

Here, the group is being asked to question the meaning of the sinusoidal data that they have visualized graphically. Given the focus on interpreting data in Weintrop et. al's definition of investigating systems as a whole, we expect to see this practice.

Overall, we wanted students in P^3 to be able to make sense of/interpret the data generated by a system as it pertains to the phenomenon under investigations, in addition to how it pertains to the underlying model.

The practice of **understanding relationships** in a system involves *identifying* the individual elements of the system and *explaining* their relationships to one another. For example, a group might identify the mass and local acceleration due to gravity as the individual elements of the system of the local gravitational force. The group could then explain the relationship between these two elements as they relate to the force (i.e., the force is propor-

tional to both the mass and acceleration). This type of practice was observed 13 times across the data, accounting for 13% of the systems thinking practices, and 4% of all practices.

Understanding relationships most likely shows up in our data given the learning goal of being able to develop and use a model (P1). Developing a model involves an iterative process of creating the model, making predictions with it, and validating the model based on its results [1]. Throughout this process, the individual elements of the system must be identified and correctly related to one another. For example, consider Excerpt 23 from Group H where they are validating their model based on the relationship between the force and the separation distance:

SC: So right now the force is just always acting in this way in the negative x-direction.

SC: But the force needs to decrease as we get farther away...

SB: Okay...

SC: So if we put the satellite dot position down here...

SC: We could make it do that...

SA: Right so let's do that then.

Here, the group has identified the individual elements of the force and the position of the satellite. Further, they are explaining the way in which the two should be related (i.e., and inverse dependence). Given these types of interactions, we expect to observe this practice in our data.

Additionally, understanding relationships likely shows up in our data given that the course is designed to cover multiple force models with widely varying complexity. Specifically, the first week focuses on a constant velocity motion, with a relatively simple constant zero

force model. The second week focuses on constant acceleration motion, with a slightly more complicated constant local gravitational force model. The third week focuses on non-constant forces, like the centripetal force and the Newtonian gravitational force, which are general, complex, and difficult to grapple with. Given the complexity of the models used in the third week, it takes a significant amount of time and discussion to develop a strong understanding – which we can then observe.

Overall, we want students in P^3 to understand that different mathematical relationships result in different types of physical behavior.

6.1.3   Unobserved practices

Six of the practices laid out by the framework have not been observed at all. Accordingly, these practices (i.e., collecting data, manipulating data, choosing computational tools, developing modular solutions, preparing problems, defining systems) are given their due attention. Although these practices are unobserved, it is still important to discuss why they are unobserved.

The practice of **collecting data** is not expected to show up in our data given that there are no sensors or meters that are provided to students, as they might be in a lab setting. The only tool students are required to use is the computer along with VPython. This tool, although it can be used to handle the collection of data, is primarily meant to be used to create the data algorithmically. This aligns with the learning goals of developing and using models (P1) to create data, rather than collecting it.

Overall, we want students in P^3 to be able to create large and complicated sets of data using various models and algorithms, rather than to simply collect it with a sensor.

The practice of **manipulating data** is not expected to show up in our data given that

its definition (see Sec. 2.1) focuses on the reshaping of data (e.g., filtering a set of data or merging two sets of data into one). Students are not required to reshape data in this way in P^3 (e.g., by using the pandas package to merge two data sets). Rather, they are required to create the data algorithmically, which can then be visualized or analyzed.

Any manipulation of data, in its most generous sense, happens at the level of the model or algorithm that is creating the data. Accordingly, excerpts that might generously be considered manipulating data are better classified as creating data (see Sec. 5.2.1).

Overall, we don't expect students in P^3 to be able to reshape/clean-up large of sets of data, rather, we want them to be able to correctly create those large sets of data using mathematical and computational models.

The practice of **choosing computational tools** is not expected to show up in our data given that the tool they are required to use it provided to them. The first three MWPs are all implemented through VPython and require no additional tools. Accordingly, by the third problem, students are familiar with the tool and know to take advantage of it.

It should be noted that nothing precludes students from using other tools (e.g., Microsoft Excel) to solve the problem, however we have not observed this in our data. This is likely due to the lack of a learning goal focusing on tool selection.

Overall, we want students in P^3 to become proficient with the tool of VPython for modeling motion, rather than being able to choose between competing tools.

The practice of **developing modular solutions** is not expected to show up in our data given that the computational problems from week to week are sufficiently different that new models must always be used. This does not allow for much cross-over or reuse between solutions. Specifically, the first problem involves no net force, the second problem involves a piecewise constant net force, and the third involves a non-constant net force – all

being sufficiently different to warrant the construction of unique computational models. This repeated design, construction, and assessment of new models, written from scratch, aligns well with the learning goal of developing and using models (P1).

Overall, we students in P^3 to be able to design, construct, and assess new models from scratch, rather than be able to reuse old models.

The practice of **preparing problems for computational solutions** is not expected to show up in our data given that the problem has already been cast in a form that is amenable to a computational solution. In fact, this is the third problem that they have seen like this, so they already know to approach it computationally.

Any preparation of a problem, in its most generous sense, happens at the design stage. Accordingly, excerpts that might generously be considered preparation of a problem are better classified as designing a computational model (see Sec. 5.2.3).

Overall, we don't expect students in P^3 to generate their own problems (aside from the create your own problem day...), rather, we expect them to be able to solve well-defined problems.

The practice of **defining systems and managing complexity** is not expected to show up in our data given that most students have very little prior programming experience. This is possibly due to the fact that there are no computational prerequisites for P^3. Given this lack of prior programming experience, interaction with the programming system as a whole is restricted by design. Although there is an instructor generated system that students are using (i.e., a MWP in Python that interfaces with PhysUtil and the Visual module), its complexity and management are beyond the scope of the course. This is reflected in the absence of a learning goal (see Sec. 3.4) focusing on the system as a whole. Additionally, the problem statement itself does not explicitly require students to interact with the system as

a whole, and tutors will dissuade this action.

Further, the first three MWPs all follow the same basic program structure: using a single calculation loop to integrate Newton's equations of motion with a particular force model. Accordingly, the vast majority of time is spent working on the force model, rather than engaging with the system itself. In this way, we limit the students' interactions with defining systems and managing their complexity.

Overall, we don't expect students in P^3 to understand or modify the underlying system itself, rather, we just expect them to be able to use it to solve the problem.

6.2 Limitations/Constraints/Considerations?

Although the framework gives a solid foundation and a good starting place, it does not come without its limitations. Additionally, the course and its design constrains the practices that we observed. In this section we describe the limitations of the framework and the constraints of the course, and discuss some of the improvements that can be made.

6.2.1 Framework

When identifying the practices laid out by the framework, many unanswered and difficult questions invariably arise. This is mostly because there is a fair amount of overlap between some of the categories and practices, making it difficult to clearly identify separate practices. Additionally, some of the definitions are rather vague and ambiguous. In what follows, we describe the difficulties with each category and provide discussion on the ways to improve the framework.

6.2.1.1 Data

Visualizing data has a tendency to cooccur with the practice of analyzing data. This cooccurrence might be expected given that students often analyze data by visualizing it. Either way, we need to be clear about defining the fundamental difference between the two: analyzing data occurs at the level of the individual data points (e.g., looking for a pattern in a set of numerical values), whereas visualizing data occurs one level removed (e.g., visualizing the slope of a 2D graph). Although these practices do commonly occur together, they need not, and so care must be taken to correctly identify each.

6.2.1.2 Modeling

Finding and testing solutions was difficult to apply within our data given its overlap with assessing computational models (see Sec. 5.2.4). This is especially true within our data, given that the solution being tested is a model being assessed. In both cases, there is some form of assessment or testing. These two could be distinguished by clearly defining what is meant by a "model." If that is not possible within a particular discipline, then these two really should be combined into one practice.

Additionally, understanding concepts and communicating information are expected to commonly cooccur. This is because the understanding of concepts needs to be communicated, verbally or otherwise, in order for us to be able to observe it. That is, sometimes when students are communicating information, they are communicating their understanding of different physical concepts. This overlap could be separated by more clearly defining what constitutes an "understanding." If that is not possible within a particular discipline, then these two really should be lumped into one.

125

6.2.1.3 Problem solving

Assessing solutions is expected to show up less frequently within our data given its similarity to assessing computational models (see Sec. 5.2.4). This is especially true in our data set, given that the solution being assessed is model. Aside from the relatively rare cases where there is an explicit comparison between two different models, most instances of assessment are better classified as assessing models. These two could be distinguished by more clearly defining what a "solution" consists of. If that is not possible within a particular discipline, then these two really should be combined into one practice

6.2.1.4 Systems

Understanding relationships in a system has possibly shown up less commonly in our data given that the practice is ambiguously defined – it can have a significant amount of overlap with the practice of designing a computational model (see Sec. 5.2.3). This overlap ultimately depends on the ambiguous definition of a system given by Weintrop et. al. For example, if a computational force model can be considered a system, then anytime students are designing a computational model they are also understanding the relationships in a system. Rather, if a system refers to something more like a collection of files that are related to one other to create a program (e.g., a Python script that loads different modules/libraries), then understanding the relationships in a system would most likely not occur along with designing a computational model. This type of ambiguity has a tendency to lower confidence during inter-rater reliability. Accordingly, this practice is confidently observed less often.

Further, understanding concepts is expected to show up less frequently within our data given the sheer research difficulty of defining and observing understanding. Aside from relatively rare cases where groups explicitly articulate their perceived understanding of a par-

ticular concept (see communicating information of Sec.), this practice often goes unnoticed, regardless of whether or not understanding is actually taking place.

6.2.2 Context

In any classroom, there are many design decisions that either encourage or do not encourage practices. For example, deciding whether to have a group or individual approach. Or, deciding whether to have guided vs. un-guided instruction. There are many practices that are explicitly encourage in P^3, but some are not encourage enough.

Given the design of the course and the limitations of the programming needed, we only see a limited set of instances of programming. However, more sophisticated classrooms requiring more complicated models might yield new and interesting instances of these practices.

6.3 Opportunities

6.3.1 Research

6.3.1.1 Productive practices

6.3.1.2 Individual vs. group

6.3.1.3 Extending to upper-level

6.3.2 Teaching

Describe the following sections.

6.3.2.1 Instructor preparation

6.3.2.2 Course design

6.3.2.3 Assessment

Chapter 7

Conclusion

APPENDICES

Constructing computational models

Programming

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Undergraduate Curriculum Task Force AAPT. Recommendations for computational physics in the undergraduate physics curriculum. Technical report, AAPT, 2016.

[2] Marzieh Ahmadzedah, Dave Elliman, and Colin Higgins. An analysis of patterns of debugging among novice computer science students. In *ITiCSE Proceedings*, 2005.

[3] Alfred Aho. Computation and computational thinking. *The Computer Journal*, 2012.

[4] C. Antaki, M. Billig, D. Edwards, and J. Potter. Discourse analysis means doing aanalysi: a critqiue of six analytic shortcomings. *DAOL Discourse Analysis Online*, 2002.

[5] J. Aronson. A pragmatic view of thematic analysis. *The Qualitative Report*, 1995.

[6] M. Belloni and W. Christian. Time development in quantum mechanics using a reduced hilbert space approach. *American Journal of Physics*, 2008.

[7] Virginia Braun and Victoria Clarke. Using thematic analysis in psychology. *Qualitative Research in Psychology*, 2008.

[8] A. Buffler, S. Pillay, F. Lubben, and R. Fearick. A model-based view of physics for computational activities in the introductory physics course. *American Journal of Physics*, 2008.

[9] Alan Bundy. Computational thinking is pervasive. *Journal of Scientific and Practical Computing*, 2007.

[10] Marcos Caballero, Matthew Kohlmyer, and Michael Schatz. Fostering computational thinking in introductory mechanics. In *PERC Proceedings*, 2011.

[11] Richard Catrambone. The subgoal learning model: Creating better eexample so that students can solve novel pproblem. *Journal of Experimental Psychology*, 1998.

[12] Ruth Chabay and Bruce Sherwood. Computational physics in the introductory calculus-based course. *American Journal of Physics*, 2008.

[13] B. Chandrasekaran. Design problem solving: a task analysi. *AI maganize*, 1990.

[14] Norman Chonacky and David Winch. Integrating computation into the undergraduate curriculum: A vision and guidelines for future developments. *American Journal of Physics*, 2008.

[15] D. Cook. Computation in undergraduate physics: The lawrence approach. *American Journal of Physics*, 2008.

[16] B. Crandall, G Kelin, and R. R. Hoffman. *Working minds: a practioner's guide to cognitive task analysis.* Massachusetts Institute of Technology, 2006.

[17] A. A. DiSessa and H Abelson. Boxer: A reconstructible computerional medium. *Communications of the ACM*, 1986.

[18] F. Esquembre. Easy java simulations: An open-source tool to develop interactive virtual laboratories using matlab/simulink. *International Journal of Engineering Education*, 2005.

[19] Jennifer Fereda and Eimear Muir-Cochrane. Demonstrating rigor using thematic analysis: A hybrid approach of inductive and deductive coding and theme development. *International Journal of Qualitative Methods*, 2006.

[20] Sue Fitzgerald, Gary Lewandowski, Renee McCauley, Laurie Murphy, Beth Simon, Lynda Thomas, and Carol Zander. Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers. *Computer Science Education*, 2008.

[21] Bent Flyvbjerg. Five misunderstandings about case-study research. *Qualitative Inquiry*, 2006.

[22] Shuchi Grover and Roy Pea. Computational thinking in k-12: A review of the state of the field. *Educational Resarcher*, 2013.

[23] N. T. Hawkins, P. W. Irving, and M. D. Caballero. Understanding student perceptions of computational physics pproblem in introductory mechanics. In *PERC Proceedings*, 2017.

[24] Wm. G. Hoover and C. G. Hoover. Computational physics wth particles. *American Journal of Physics*, 2008.

[25] P. W. Irving and E. C. Sayre. Developing physics identities. *Physics Today*, 2016.

[26] Paul Irving, Michael Obsniuk, and Marcos Caballero. P^3: A practice focused learning environment. *European Journal of Physics*, 2017.

[27] H. Joffe and L. Yardley. *Research Methods for Clinical and Health Psychology.* Sage, 2004.

[28] B. Kirwan and L. K. Ainsworth. *A guide to task analysis.* Taylor & Francis, 2005.

[29] Matthew Kohlmyer. *Student performance in computer modeling and problem solving in a modern introductory physics course.* PhD thesis, Carnegie Mellon University, 2005.

[30] Renee McCauley, Sue Fitzgerald, Gary Lewandowski, Laurie Murphy, Beth Simon, Lunda Thomas, and Carol Zander. Debugging: a review of the literature from ane ducational perspective. *Computer Science Education*, 2008.

[31] D. H. McIntyre and C. A. Manogue. Integrating computational activities into the upper-level paradigms in physics curriculum. *American Journal of Physics*, 2008.

[32] Committee on a Conceptual Framework for New K-12 Science Education Standards. *A Framework for K-12 Science Education: Practices, Crosscutting Concepts, and Core Ideas*. The National Academies Press, 2012.

[33] Seymour Papert. Teaching children thinking. *Mathematics Teaching*, 1972.

[34] Seymour Papert. *Mindstorms: Children, computers, and powerful ideas*. Basic Books, 1981.

[35] Seymour Papert. An exploration in the space of mathematics educations. *Technology, Knowledge, and Learning*, 1996.

[36] Katherine Perkins, Wendy Adams, Michael Dubson, Noah Finkelstein, Sam Reid, Carl Wieman, and Ron LeMaster. Phet: Interactive simulations for teaching and learning physics. *The Physics Teacher*, 2006.

[37] Jean Piaget. *Origins of intellifence in children*. W. W. Norton and Company, 1963.

[38] J. Potter and M. Wetherell. *Discourse analysis as a way of analysing naturally occurring talk*. Sage, 1997.

[39] Edward Redish and Jack Wilson. Student programming in the introductory physics course: M.u.p.p.e.t. *American Journal of Physics*, 1992.

[40] David Sherer, Paul Dubois, and Bruce Sherwood. Vpython: 3d interactive scientific graphics for students. *Computing in Science & Engineering*, 2000.

[41] M. Vaismoradi and T. Bondas. Content analysis and thematic analysis: implications for conducting a qualitative descriptive study. *Nursing and Health Sciences*, 2013.

[42] L.S. Vygotsky. *Mind in Society: The Development of Higher Psychological Processes*. 1980.

[43] Shawn Weatherford. *Student use of physics to make sense of Incomplete but functional VPython programs in a lab setting*. PhD thesis, North Carolina State University, 2011.

[44] C. E. Weiman, K. K. Perkins, and W. K. Adams. Interactive simulations for teaching physics: what works, what doesn't, and why. *American Journal of Physics*, 2008.

[45] David Weintrop, Elham Behesthi, Michael Horn, Kai Orton, Kemi Jona, Laura Trouille, and Uri Wilensky. Defining computational thinking for mathematics and science classrooms. *Journal of Science Education Technology*, 2015.

[46] S. Widdicombe and R. Wooffitt. *The language of youth subcultures: social identity in action*. Harvester Wheatsheaf, 1995.

[47] Jeanette Wing. Computational thinking and thinking about computing. *The Royal Society*, 2008.

[48] Jeannette Wing. Computational thinking. *Communications of the ACM*, 2006.