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

2018

ABSTRACT

IDENTIFYING COMPUTATIONAL PRACTICES IN INTRODUCTORY PHYSICS

By

Michael J. Obsniuk

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

TABLE OF CONTENTS

LIST OF TABLES

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 – highlightning 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 moreso.

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, 28]. Visual packages such as VPython or Glowscript [39] 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 [31]. 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 [47]."

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 [44]. 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

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, 31, 44], 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 [33, 34, 47, 46, 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 [36]. 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 [33].

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 [32]. 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 [33]." Papert would later go on to say that computational thinking involves "forging new ideas" that are both "accessible and powerful [34]."

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. [47]"

7

Wing is careful to remind readers that computational thinking is a fundamental skill for everyone, not just computer scientists [46]. 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 [31]:

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, 38, 14, 30], the results from DBER (e.g., student challenges) [12, 8, 6, 23, 15, 43], 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 [39]."

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
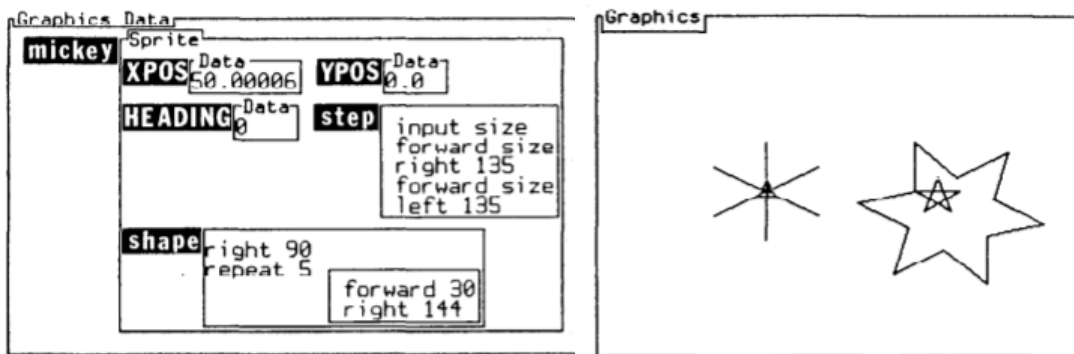
Figure 2.1: Graphical user interface for BOXER showing the graphics data (e.g., x-position, y-position, step instructions) and the resulting graphic for a sprite named mickey.

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 [42].

Another implementation of computation, frequently used today, are the Physics Education Technology (PhET) simulations [35]. 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

12

```
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 [32, 17, 35, 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., $d\vec{p} = \vec{F}_{\text{net}} \, dt$ 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 [28]. 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 [42]. 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

17

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 profficiency, 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 Newtonian 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 appicability 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 defintions 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 computationl 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 defintions 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 [27]. 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 begining 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 heirarchy* 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

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, 40, 26, 37, 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" [24]. 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 [37, 45]. 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. 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

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

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),

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) [41]. 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,

P6. constructing explanations,

P7. engaging in argument from evidence.

P8. 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 [25] 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).

38

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. The following chapter was 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. 4.1.1.2, 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.

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 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. [29]" 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 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
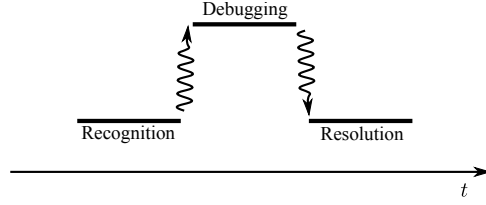
Figure 4.1: The debugging process necessarily corresponds to a phase beset on either side by the phases of recognition and resolution. Note the absence of a vertical scale, as the vertical separation merely acts to distinguish phases.

on either side of the process of *physics debugging*, as represented in Fig. 4.1. 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

4.1.1.2 Resolution

Chapter 5

Observations

Throughout this thesis we have made many different types of observations, and have used those observations to help answer many important research questions (see Ch. 2). Accordingly, it is important that we take some time to detail the process of and results from those observations. In this chapter, we detail the methodology of our analysis (i.e., the process) and illustrate the identification of three of the most frequent practices (i.e., the results): troubleshooting and debugging, assessing computational models, and creating computational abstractions.

5.1 Analysis

Our full analysis involves many different stages: first the inital data was collected and susequently reduced in order to provide a manageable set of data, second an independent coding scheme was generated – using the Weintrop framework – to help identify computational practices, and third multiple inter-raters were used to ensure reliability of the analysis. Each of these stages are detailed below.

5.1.1 Data reduction

Following the suggestions of thematic analysis, we began with a full transcription of the in-class video. Our video data was collected from nine groups working on three different but related computational problems (see Ch. 3). Slowly, we found that the most interesting things were occurring in the last (and most complicated) of the computational problems. Thus, we focused our analysis on the satellite problem.

Each transcript has been transcribed verbatim 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 researcher (e.g., referring to a previous equation), we follow the convention of square brackets ([]) and curly brackets ({ }), respectively. For example, Fig. 5.1 shows a portion of transcript.

Each full transcript is read through multiple times, noting any points of interest in terms of the Weintrop framework for computational practices involving computational thinking [44]. We then divide the transcript into multiple excerpts of shorter length centered around those initial points of interest. Although we lose some of the coherence of the problem as a whole by focusing on shorter excerpts, these relatively shorter excerpts help to facilitate a deeper analysis. Each transcript contains ≈ 1500 lines of speech comprising ≈ 50 excerpts.

A preliminary analysis of each transcript was done with an eye towards finding candidate excerpts where students were i) solving the satellite problem and ii) where they were using the computer. There is a fair amount of off-topic conversation in each transcript, which can easily be identified and removed without worry. However, a closer look is required to make sure that we are not throwing away any valuable data.

This closer look is what ultimately helps us to more clearly define what a computational

112	SA	SB	SC	SD	TA
113			so we all agree that we change the mass of the satellite		
114	are you sure thats the mass?				
115				isnt this the mass, you would think?	
116			where is it...		
117	times ten to the third				
118				are you supposed to have a space in between?	
119			i dont think so		
120	oh wait, its star star				
121	i think				
122		no that was to do a power			
123	its ten to the third				

Figure 5.1: A portion of transcript meant to highlight the indication of unspoken and inferred actions.

112	SA	SB	SC	SD	TA
113			so we all agree that we change the mass of the satellite		
114	are you sure thats the mass?				
115				isnt this the mass, you would think?	
116			where is it...		
117	times ten to the third				
118				are you supposed to have a space in between?	
119			i dont think so		
120	oh wait, its star star				
121	i think				
122		no that was to do a power			
123	its ten to the third				

Figure 5.2: A portion of transcript.

practice both is and is *not* within an excerpt. By removing the off-topic and otherwise irrelevant information, we are focusing our analysis specifically on group computational problem solving and its associated practices. For example, Fig. 5.2 shows a portion of transcript containing multiple candidate excerpts with some off-topic conversation.

Initially, we used the various computational practices already defined by Weintrop, along with their associated characteristics, to identify a reasonable amount of candidate excerpts. Each candidate excerpt is then scrutinized individually and as compared to other excerpts to more clearly define the characteristics within the data – the more characteristics observed, the more likely that what we are looking at really is a particular computational practice. **In this way, we reduce our data to a manageable and relevant set.**

5.1.2 Coding process

Each excerpt is analyzed in terms of the practices the group is engaging in. This means we need to look for the characteristics of each practice, as described in Sec. 2.1, to give us an initial footing. With these characteristics identified in the data, we are justified in classifying

51

SA*	SB	SC	SD	TA	Tags	Reason	Reason within Excerpt	Reason beyond Excerpt
1								
ill write down all our numbers [on the board] so we dont have to [mumbles]					Defining systems and managing compelxity	The group is defining the "boundaries" of a "system" to investigate a "question."	The "boundaries" of the system at this point are the mass of the satellite (line 82) and the circumference of the orbit (line 85). The "system" is the program as a whole.	The "question" they are investigating is how they can modify the program to realistically simulate the trajectory of the satellite in geostationary orbit.
i dont remember what...					Analyze data	The group is starting to analzying a set of "data" to make "conclusions."	The "data" consist of the mass of the satellite (line 82) and the circumference of the orbit (line 85).	The "conclusion" that they eventually come to to to change the mass of the satellite in the program (line 103).
			facts, lacking, assumptions, and ..		Preparing problems for computational solutions	The group is "reframing" a "problem" so that "progress" can be made with computation.	The group is "reframing" the problem by attaching all of the known/given information to variables (line 68). The "progress" that is made is that they define these variables in code (line 82).	The "problem" is to correctly simulate the trajectory of a geostationary satellite.
do we need all of those today?								
		uh, i dont know, probably not, but you can still write em...						
			You know when we got like the distance from Earth?					

Figure 5.3: Visual representation of the different levels of rationale pertaining to an excerpt. Green, yellow, and red cells represent high, medium, and low confidence, respectively.

the particular excerpt as that practice. Each excerpt generally has between one and four identified computational practices, and each practice generally has between three and eight identified characteristics.

For example, Fig. 5.3 shows Excerpt 42 from Group H where a few characteristics have been identified. These four characteristics, as further analysis will show, pertain to the computational practice of creating and visualizing data.

Each characteristic that is identified is supported with rationale on three levels: the rationale according to the framework, the rationale within the excerpt, and the rationale beyond the excerpt. For example, the rationale beyond an excerpt might pertain to a relevant statement made minutes (sometimes hours) before an excerpt. The rationale within an excerpt usually references a line (or few lines) of the associated transcript. These rationale are then used to support the claim that a number of characteristics have been identified. These characteristics are listed under the rationale according to the Weintrop framework.

The result of this process is illustrated in the excerpt shown in Fig. 5.3.

The confidence in each rationale (specifically, the rationale within the excerpt) manifests itself in a high, medium, and low confidence rating for the identification of this practice. In that way, we know not only how frequently the practices are identified, but also how confident we are in that assessment. The result of this process is illustrated in the excerpt shown in Fig. 5.3.

5.1.3   Inter-rater reliability

We indicated confidence by marking a potentially identified practice as either high (green), medium (yellow), or low (red) confidence. To determine confidence, a process of inter-rater reliability was conducted between three raters. This process included sharing a representative sample of the excerpts and converging on the actual practices observed. Rationale for every high- to medium-confidence practice, then, is provided according to the framework, according to the data within the excerpt, and according to any data beyond the excerpt.

With these practices identified, we then analyzed their frequency. Many practices showed up frequently, and some showed up almost never. Analysis of these simple statistics, represented visually in Fig. 5.4 give us a concrete way to hone in and focus our analysis on the most important (or at least frequent) practices. The most important practices were then further analyzed for the variation in the characteristics that make up each practice.

So that we safeguard against the researcher influencing the results too much, inter-rater reliability is checked at multiple points throughout the analysis. This check consists of an independent researcher (the inter-rater) identifying practices with confidence in a reduced set of the data and comparing with the primary researcher. Any practices in which confidence is questioned and lost are discounted from being identified.

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 5.4: A "density plot" for all computational practices and all groups, showing that many practices show up frequently, while other show up never.

5.2 Computational practices

We have identified a number of frequent computational practices within our data set.

5.2.1 Troubleshooting and debugging

In Excerpt 3 from Group D, a group can clearly be seen engaging in the practice of troubleshooting and debugging. This practice usually requires that students identify an unexpected error in the program and work to systematically rectify it. For example, SC can be seen identifying an error:

SC: try it...

SC: im not sure it gonna run [inaudible]

SC: type is wrong [reading error statement]

as she reads the error output from the VIDLE shell. This error output reads: `Traceback (most recent call last):   File "satellite.py", line 16, in <module>`. Given the output of the VIDLE shell, SC identifies that the unexpected error that they have run into is that they are trying to use a scalar for something that needs to be defined as a vector. Once this error has been identified, the group begins to systematically work towards rectifying it.

Accordingly, the group can be seen going back through their code and systematically checking for variables that can be turned into vectors:

SB: it needs to be a vector

SC: yeah it should be vector type but it wasn't...

SC: so the problem is, this one [r] is vector, this one is not a vector [m]

Tag	Rationale from framework	Rationale within excerpt	Rationale beyond excerpt
Troubleshooting and debugging	The group has **identified** an **unexpected problem** and working to correct it in a systematic manner.	The group has **identified** a problem through the output of VPython error shell (line 57). The **unexpected problem** is that they have defined their force as a scalar but it needs to be given a direction (line 67).	

Figure 5.5: Caption.

SC: the problem is not a vector and vector, we trying to combine together

SC: we never gonna get answer as computer cause its a computer

Mistakenly, SC believes that in order for the program to be able to calculate their net force, they need to turn every other variable used in the calculation of the net force into a vector (e.g., the mass of the satellite).

Subsequently, Group C can be seen systematically working toward rectifying that error by turning each variable used in the calculation of the net force into a vector. Although this group is later encouraged by the TA to change some of the variables back to scalars, this Excerpt, summarized in Fig. 5.5, is a clear example of a systematic process of troubleshooting and debugging.

5.2.2 Assessing computational models

The practice of assessing a computational model manifests itself in Excerpts 6 and 7 from Group D. With these excerpts, we are able to justify *that* the Group D is engaging in assessing

a computational model through all of Excerpt 6, and we are able to explain *why* they are engaging in that practice through most of Excerpt 7. These excerpts are presented in detail below.

5.2.2.1 Justification

Excerpt 6 shows the group, broadly speaking, as they modify their net force so that it involves a vector position as opposed to just its magnitude. They begin by first considering the general modification:

SC: oh no its infinitely running

SC: and error is again, oh gee

SA: maybe we have r defined as the distance that we calculated last time and maybe we should have r defined as a formula or a vector... r needs to be a vector.

SB: you agree with that SC?

SB: that r should be defined as like a formula or a vector instead of number?

SC: you guys change, you talk about r changing as vector type

SA: yeah

SC: how?

That is, we see SA questioning whether or not they should be using the distance as a numerical scalar as opposed to a formula or a vector. Ultimately, he decides that they should be using a vector. The group is not ultimately sure if this is what they should be

doing, but SB asks SC if she agrees with SA, and they at least begin to questions how to implement the change.

Next, they start to go about implementing a change by deleting what they have previously defined as the distance that they calculated last time (i.e., D=1E6). This distance represents the radius of the roughly circular orbit of the satellite in geostationary orbit:

SD: the position depends on the earth radius...

SC: oh can we just do the... oh they already give us earth radius

SC: so?

SB: so...

SD: delete that [D]

SC: just delete?

SD: and then...

SB: do i delete that?

SD: yeah yeah

SC: D part?

SC: and then we put the r instead of what we had there

Not only does the group decide to delete what they have previously defined, but they then discuss what should be added in its place. Although they aren't exactly sure *how* to replace the scalar D with the vector r, they at least know that it must be done.

Finally, they begin to replace what hey have deleted with a vector position of the satellite as `satellite.pos`:

SB: what do you want there in place of D?

SC: r, is the radius for...

SA: the magnitude of R needs to be distance

SC: no no just wrote down the satellite

SB: oh you just want me to put satellite?

SC: satellite dot pos

SB: satellite... [typing]

SC: yeah dot position

It takes a little figuring to work out that instead of defining a new variable r as a vector, they can just directly use the pre-defined `satellite.pos`. This vector r is what eventually gets used in the construction of the Newtonian gravitational force – specifically, to give it a direction.

Again, this excerpt can be used to justify *that* the group is modifying their net force – in other words, that they are engaging in the practice of assessing a computational model. Specifically, as summarized in Fig. 5.6, the model that they are assessing is a Newtonian gravitational force that is a scalar as opposed to a vector (i.e., a force that has a direction). This comparison between what the model is and what the model should be is one of the hallmarks of the practice of assessing a computational model.

Tag	Rationale from framework	Rationale within excerpt	Rationale beyond excerpt
Assessing computational models	Identify model, phenomenon, and comparison.	Here the group is clearly articulating THAT they need to change the type of variable that represents the position of the satellite (scalar to vector). Ultimately, this is the comparison that they are making -- between the scalar B and the vector r.	The phenomenon is a a central force.

Figure 5.6: Caption.

5.2.2.2 Explanation

Excerpt 7 shows the group beginning to construct a Newtonian gravitational force in their code that specifically takes a direction into account through a unit vector. They first begin by recognizing that the force needs to have a direction:

SA: yeah i dunno {if the mass needs to be defined with the realistic value}

SA: great question... [calculates the magnitude of the force on his calculator]

SC: yeah just try that one [points to $\vec{F} \propto \hat{r}/r^2$] first

SD: but the direction of F is...

SD: so we need to turn the r into a vector

This recognition is implied by SD when she mentions a "direction of F" and then suggests that they turn their previously defined radius of the roughly circular orbit r into a vector to rectify the mistake. The implication being that by defining their net force in terms of a vector position, it (the force) will take on the desired direction.

Tag	Rationale from framework	Rationale within excerpt	Rationale beyond excerpt
Assessing computational models	Identify model, phenomena, and comparison.	Here the group is clearly articulating WHY they need to change the type of variable that represents the position of the satellite is -- because the force needs to have a direction like the position does. The group begins to re-define their net force in terms of the vector position of the satellite so that it picks up this vectorial nature.	The phenomenon is a central force.

Figure 5.7: Caption.

With the recognition that the net force needs to be a vector (i.e., have a direction), the group then tentatively decides on an equation that they should be trying to replicate in code:

SC: i think we should...

SD: [writes $\vec{F} \propto \hat{r}/|\vec{r}|^2$]

SD: so just put the `r` value, vector value and just put this uhh {mag()} function

SB: as a parameter?

This equation specifically takes the direction into account by using the vector position of the satellite to calculate a unit vector as well as its magnitude by the `mag()` function.

Again, this excerpt can be used to explain *why* the group is modifying their net force – that is, why they are engaging in the practice of assessing a computational models. Specifically, as summarized in Fig. 5.6, the group begins to change the way that the net force is defined.

5.2.3 Computational abstractions

Sometimes, it is easier to describe what a practice is *not*, rather than what it is – though both descriptions are of interest and should be taken together. For example, we can compare Excerpts 3 and 4 from Group D as they exemplify both of these types of descriptions – giving us a well-rounded description of what it means for a group to create a computational abstraction.

5.2.3.1 What it is not

Excerpt 3 shows the group identifying a computational abstraction as they work to advance their scientific pursuit of generating the trajectory of a geostationary satellite. The begin by recalling a point in time (i.e., the last computational problem that they worked on last week) where they were provided with a "dummy value" that needed to be changed to a more realistic one:

SD: yeah `Fnet`

SC: but i dont know where the five thousand came out of

SB: well remember the one time we just started and it {the program} like gave us ten or
something?

SC: oh just... they are just giving?

SC: okay...

Here, the group can be seen referencing the mass of a hovercraft as it pre-defined in the second computational problem that each group worked on together. Eventually, they had

to change this pre-defined 'dummy' value to the realistic value for the particular situation. This is very similar to what is actually going on in this excerpt.

The group has first recognized that the pre-defined momentum of the satellite is not something that they are completely sure of. Once the group has recalled that something like this has happened before, they then suggest that they should take their more realistic numerical values and use them to calculate the force:

SA: what if we just put this [points to mass of satellite] in here [points to `m` in `G*m*M/D**2`]?

SC: yeah

SA: and we put this [points to the radius of geostationary orbit on the WB] in there [points to `D` in the equation `G*m*M/D**2`]

SA: would that give us our force?

In this way, the group is modifying their code and advancing their scientific pursuit of generating a more realistic simulation. Ultimately, these modifications were due to the scrutiny of the pre-defined numerical value of a variable.

Although this interaction, as summarized in Fig. 5.8, seems to involve a computational abstraction in some loose sense, it is actually more useful as an example of what a computational abstraction is *not*. That is, although the group has identified a variable that is defined using a "dummy" value and begin to modify them to reflect the realistic situation, they have not really created a computational abstraction. The creating of a computational abstraction, then, seems to be more of a complicated procedure, as shown below.

Tag	Rationale from framework	Rationale within excerpt	Rationale beyond excerpt
Creating computational abstractions	The group has **identified** a computational **abstraction** as they advance.	The **abstraction** they are working with is for the mass of the satellite (line 93). They have **identified** this abstraction through its name (line 94).	This is a good example of what computational abstractions are NOT. There is an important difference between scrutinizing the numerical value of an abstraction (what this group is doing) and creating a computational abstraction.

Figure 5.8: Caption.

5.2.3.2 What it is

Excerpt 4 shows a more clear-cut example of what creating a computational abstraction looks like. The group begins with SD making a suggestion to define a function for the net force, as opposed to entering a single numerical value:

SD: so just put the r value, vector value

SD: just put this uhh function

SB: as a parameter?

SD: just give the computer a function so we dont have to calculate F {like SA is doing}

SB: thats a good idea

SB: should we put it in the parameter?

Although SB seems to agree that they should be constructing a function, she is not quite sure where to put and asks if she should place it in the parameter section.

64

Although they never finally agree on where to place the function (i.e., in the parameter section or calculation loop), they continue to discuss how they might go about constructing the function:

SB: what would it be?

SC: and what is the r value?

SD: r is the position...

SB: you calculated that {the force} out?

SA: thats what i have is the force, yeah

SB: so should i put a parameter in or...

SA: i dunno

SD: position of satellite...

SC: do they have a position of the earth

SC: and a position of the satellite?

SB: yeah here [points to code]

SA: so its saying the earth is at the origin

The group seems to be split in this portion of the transcript: one group (SA and SB) is still focusing on where to place the function and the other group (SC and SD) are discussing how to construct the function by using variables already defined in the program (i.e., the position of the Eart and the satellite). Thus, the group can be seen working to tie multiple abstractions together into a single more complex abstraction or function.

Tag	Rationale from framework	Rationale within excerpt	Rationale beyond excerpt
Computational abstraction	The group is identifying, creating, and using a computational abstraction as they work toward a goal.	The abstraction that they have identified and are working to create in code is the vector position of the satellite (as opposed to just its magnitude). This vector variable is subsequently used in the construction of their Newtonian gravitational force.	

Figure 5.9: Caption.

As summarized in Fig. 5.9, the group can be seen deciding to construct a complicated function in code, as opposed to using numerical values that they have calculated on their TI-83s. This creating of a complicated function looks to be the hallmark of the practice of creating computation abstractions. Allowing the processing power of VPython to do iterative calculations for you, by generating a complicated abstract equation, seems to be indicative of creating computation abstractions while thinking computationally.

Chapter 6

Discussion

Chapter 7

Conclusion

APPENDICES

Appendix A

Additional excerpts

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

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

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

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

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

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

[29] 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.

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

[31] 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.

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

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

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

[35] 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.

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

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

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

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

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

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

[42] 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.

[43] 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.

[44] 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.

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

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

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