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

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 [9].

Additionally, it is believed that students of computational physics gain a deeper understanding of the physical concepts [8, 14]. Visual packages such as VPython or Glowscript [19] 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 [12]. 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 [16]. 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 [4]. It is becoming increasingly clear that "computational thinking is a fundamental skill for everyone, not just computer scientists [24]."

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 [22]. This taxonomy, comprised of twenty-two individual yet inter-related practices, fitting into four different categories, is meant to help guide instructors and researchers as they attempt to teach and better understand computational thinking in science

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

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

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

4

with concluding remarks.

Chapter 2

Background

2.1   Computational thinking

As mention in the introduction, computational thinking and the associated practices are of primary interest. In order to be able to effectively and efficiently identify these practices, we need to be able to clearly identify this type of thinking.

The history of computational thinking and its defintion is long and incomplete. The term was first introduced by Seymore Papert as it related to students actively constructing knowledge (often refered to as "constructionism") through the production of an artifact (usually, but not necessarily, a computer program). However, Papert did not initially attempt to define computational thinking. Rather, he commented that attempts to integrate computational thinking into everyday life have 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 [17]." Papert would later go on to say that computational thinking involves "forging new ideas" that are both "accessible and powerful [18]."

More recently, building on Papert's preliminary observations, Jeanette Wing defines computational thinking as it relates to taking advantage of 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. Wing is careful to remind readers that computational thinking is a fundamental skill for everyone, not just computer scientists [23].

Further elaboration by Alfred Aho points out that the process of finding the right tool for the right job is a clear indicator of computational thinking. 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 [2]. 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.

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 [16]:

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.

For example, the expectation of recognizing dimensions in 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. Also, although defining computational thinking within K-12 is an ideal starting point, it must also be extended to the university level. For these reasons, 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).

2.2 Physics Education Research

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 earlier forms of computation at the introductory level, called BOXER, used "simple programming" to generate two-dimensional shapes on a computer

8
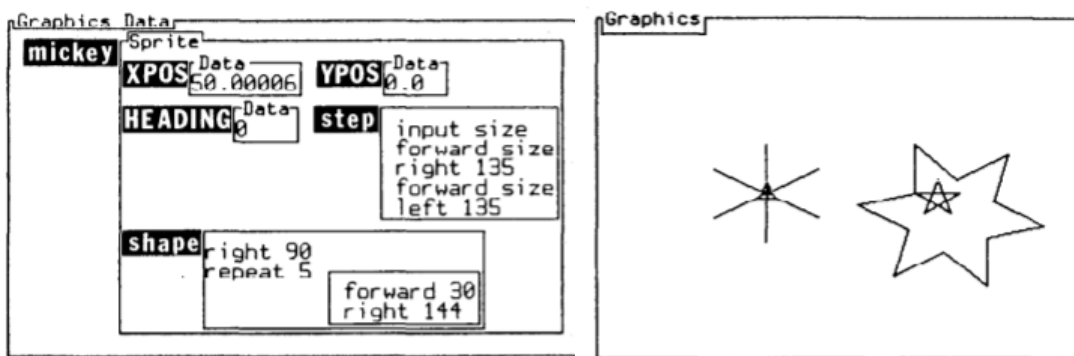
Figure 2.1: Graphical user interface for BOXER showing the graphics data and the resulting graphic.

screen. This "reconstructible medium" allowed even novice programmers to take advantage of the processing and visualization power of computers. For examples, Fig. 2.1raphical user interface for a program meant to generate a star and a triangle for 2 different objects. The underlying algorithms are laid out in sequential steps.

Alternatively, one of the more recent implementations of computation at the introductory level is called Glowscript [19]. Glowscript is a variant of VPython (i.e., the Python programming language with the Visual module), which is designed (in part) to easily generate three-dimensional visualizations. For example, the Glowscript program in Fig. 2.2 uses an Euler-Cromer integration algorithm to predict the motion of a projectile under the influence of a non-linear drag force.

This more realistic and descriptive three-dimensional visualization is thought to encourage students to form a deeper understanding of the underlying physics concepts. This and many other aspects of computation at the introductory level have been studied, from conceptual understanding to student performance.

In the early 2000s, Chabay began to research the integration of computation into the introductory calculus-based physics course [8]. This course included a computational cur-

```
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 2.2: Glowscript output demonstrating its ability to generate three-dimensional visualizations of objects, vectors, and graphs.

riculum following that of *Matter and Interactions*. Chabay found that one of the positive aspects of including computation was to stimulate creativity. This creativity in approaching problem solving is thought to lead students to the construction of more realistic computational models (i.e., taking more assumptions into account).

Around that same time, Kohlmyer dug a little deeper into student performance [14]. 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. 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 [21]. He found, among other things, that computational physics students were able to interpret physical quantities according to their variable name. For example, the mass of a satellite might be defined as `m.satellite`.

10

However, the more complicated the definition got (e.g., a function of variables like `Fnet = -k * (ball.pos-origin.pos) / mag(L)`, the more students struggled. Weatherford pointed out that their MWP activities had much room for improvement, and that more research was needed on fostering student proficiency in computational physics.

Caballero later made recommendations for integrating computation into an upper-division mechanics course [6]. He found that their model provided many affordances given its "flipped" nature. Also, focusing on the fundamental concepts allowed even introductory students to not feel overwhelmed. However, he is clear to point out that their model does not come without its shortcomings.

Caballero also suggested that investigating the types of thinking and habits of mind students develop while solving computational problems was a good place to continue study [5]. For example, Fig. 2.3 shows a program that students, under the guidance of their TAs, completed in order to generate a simulation of a satellite orbiting Earth. Based on his analysis of this problem, he concluded 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.

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 type of practices that are indicative of computational thinking.

2.3 Task analysis

Motivation for this section.

A task analysis consists of breaking a complex "task" (e.g., modeling a gravitational

```
19 while t < tf:
20
21      r = craft.pos-Earth.pos                          Force Calculation
22      rhat = r/mag(r)
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 2.3: Sample program from Caballero et. al.

system) down into related "sub-tasks" (e.g., determine the direction of the force). We aim
to identify somewhat more manageable steps that must be taken in order to complete the
overall task [7].

The task analysis of this problem was initially constructed by a single content expert.
After the first iteration it was presented to additional experts. Through this discussion, 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. 2.1 were sufficiently described to be useful in video analysis.

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 2.1: 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.

12

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 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.

2.4 Thematic analysis

Motivation for this section.

A description of the coding process, like Clarke [3]. First, verbatim transcription, making sure to indicate any gestures or implications, as well as identifying relevant writings/typings. Second, reading through the transcripts or "familiarizing" yourself with the data. Given the length of many of the transcripts, this step is necessary to break the data down into manageable chuncks. Each transcript is broken down into roughly 50 excerpts. Each excerpt contains roughly 1-3 practices that are confidently identified.

We indicated confidence by marking a potentially identified practice as either high (green), medium (yellow), or low (red) confidence. To determine confidence, however, 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. These simple statistics, like

$$\text{frequency of particular practice} = \frac{\text{number of particular practice observed}}{\text{number of all practices observed}},$$

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.

2.5 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 computational activities from general science and mathematics classrooms. Their study analyzed 32 different computational activitiesthat manifested 45 distinct practices which were classified into 4 different categories. Each practice has roughly 5 fundamental characteristics, as listed in Fig. 2.4.

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

1. Data

   (a) Collecting
      i. Propose protocols
      ii. Articulate automation

   (b) Creation
      i. Define procedures
      ii. Run simulations
      iii. Create data
      iv. Advance understanding

   (c) Analysis

   (d) Visualization

2. Modeling and Simulation

3. Problem Solving

4. Systems Thinking

Figure 2.4: 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.

itself.

As a more concrete example, the computational practice of creating data 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. Especially for the framework to become valuable to anyone looking to implement it in their classroom.

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

18

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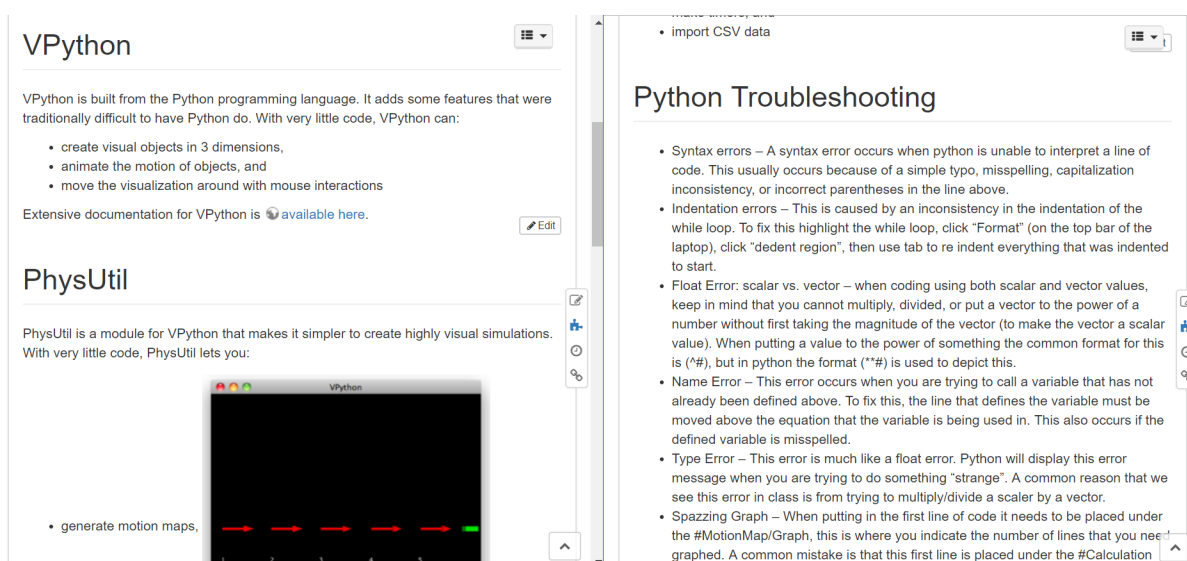
Figure 3.1: Portion of on-line notes that is made available to the students during the first week of the course. These notes introduce the fundamenatal programming ideas and a list of common errors with tips and tricks.

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),

19

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.


```
9.    Calculating a unit vector in VPython

Students in your class are continuing to model the motion of Triton (one of
Neptune's 13 moons) around Neptune, but now using VPython.  The code
your class has received contains the following snippet of VPython code.

Neptune = sphere(pos=vector(100,200,300), radius=1)
Trion = sphere(pos=vector(10,20,30), radius=2)

(a)  From this snippet, which of the following lines of code might your group
write to describe the separation vector pointing from Neptun to Triton?

  ○ rvec = Triton.pos - Neptune.pos
  ○ rvec = Neptune.pos - Triton.pos

(b)  Several groups have written different lines of code to calculate the
magnitude of the separation vector; some are correct and some are not.
From your understanding of the line(s) of code below, which of them
correctly represent the magnitude of the separation vector?

  ▢ rmag = mag(Neptune.pos) - mag(Triton.pos)
  ▢ rmag = mag(Triton.pos - Neptune.pos)
  ▢ rmag = sqrt((Triton.pos.x - Neptune.pos.x)**2
                +(Triton.pos.y - Neptune.pos.y)**2
                +(Triton.pos.z - Neptune.pos.z)**2)
  ▢ rmag = sqrt((Neptune.pos.x - Triton.pos.x)**2
                +(Neptune.pos.y - Triton.pos.y)**2
                +(Neptune.pos.z - Triton.pos.z)**2)
  ▢ rmag = mag(Neptune.pos - Triton.pos)
  ▢ rmag = mag(Triton.pos) - mag(Neptune.pos)
```

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) [20]. Constructing computational models is (unsurprisingly) a computational

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

	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 [13] 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.
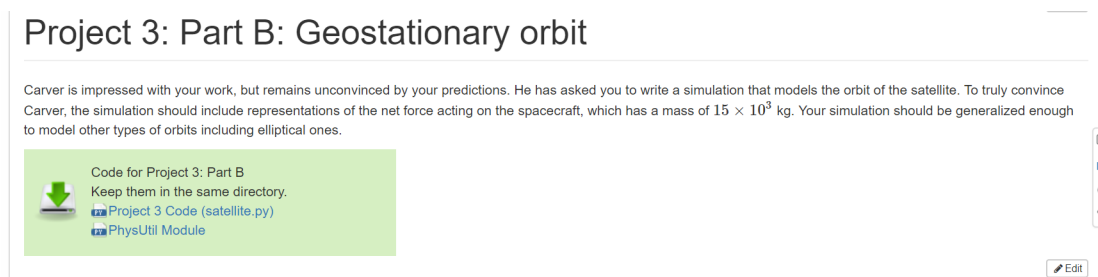

Figure 3.4: The Newtonian gravitational force problem statement delivered to the students in the third week of class.

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).

24

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

28

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

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

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 ob-

servations 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 [10]. 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. [15]" 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*.[11]

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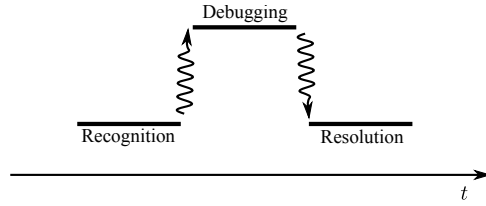
Figure 4.1: The debugging process necessarily corresponds to a phase beset on either side by the phases of recognition and resolution. Note the absence of a vertical scale, as the vertical separation merely acts to distinguish phases.

4.1.1.1 Recognition

4.1.1.2 Resolution

4.2 Phenomenography

A description of the phenomenography study.

4.2.1 Protocol

A description of the development of the protocol.

4.2.2 Analysis

Since there are no real results, we can just describe how we analyzed the interview data.

Chapter 5

Analysis

5.1   Specific methods

Following the suggestion of thematic analysis, we begin with a full transcription of the in-class video. This transcription is verbatim to the best of our abilities. Any inaudible sections are indicated, with long pauses being indicated by ellipses (. . .). To distingish between unspoken actions (e.g., pointing to an equation) and infereces made by the researcher (e.g., refering to a previous equation), we follow the convention of square brackets ([]) and curly brackets ({ }), respectively.

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

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. With these characteristics identified, we are justified in classifying the particular excerpt as that practice. Each excerpt generally has between one and four identified computational practices.

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. The confidence in this 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.

Figure 5.1: Visual representation of the different levels of rationale pertaining to an excerpt. Green, yellow, and red cells represent high, medium, and low confidence, respectively.

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.

5.2   Findings

Recall that, according to the framework, each category of practice can be broken down into a number of individual practices. Each individual practice, further, can be identified in terms of a number of fundamental characteristics. Depending on the particular situation, some categories show up in the data more often than others. For example, we expect to see fewer systems thinking practices and more computational modeling practices. Further, within any category, some individual practices show up in the data more than others. For example, within the data practices, we expect to see fewer instances of collecting data and more instances of creating data.

Most importantly, within any individual practice, there is a broad set of defining characteristics. Ideally, each characteristic would be present in some form when its corresponding practice has been identified. However, each characteristic can be organized in terms of being either necessary or sufficient for the cause. That is, we need not necessarily observe every characteristic in order to identify a practice according to the framework.
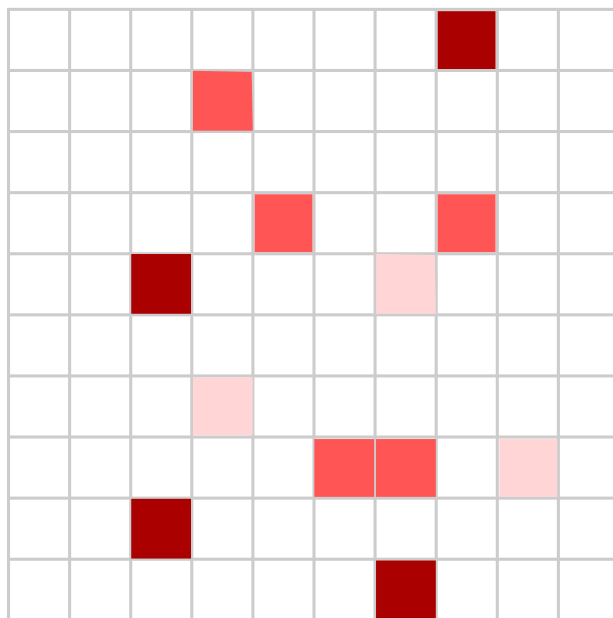

Figure 5.2: A "density plot" for all computational practices and all groups.

5.2.1   Assessing computational models

The most important computational modeling practice indicative of computational thinking is that of assessing computational models. This practice shows up $100\,\%$ of the time. Assessing computational models seems to be crucial to the process of designing and constructing computational models. Given that computational models take a lot of work and iteration to get to an acceptable level, it is no surprise that assessing shows up so frequently.

There are three characteristics that are necessary to be observed in an excerpt to warrant classification of "assessing computational models." The first characteristic that needs to be

36

observed is a "model." There are several models that show up in the data. The second characteristic that needs to be observed is a "phenomenon." For the most part, the phenomena are related to the simulation of the trajectory of the geostationary satellite and its various physical interpretations (e.g., a central attractive force or a circular orbit). The third characteristic that needs to be observed is a "comparion" between the model and phenomenon. The comparisons that we see are ultimately varied, given that they depend on not only the model but also the phenomenon.

a	b	c
c	d	$100\,\%$

Figure 5.3: A table showing rough statistics for assessing computational models.

5.2.1.1   Computational model

The two big models that we have observed in the data are a position dependent Newtonian gravitational force in terms of a separation vector and a centripetal force that depends on the polar angle of the satellite. Each model shows up $100\,\%$ and $100\,\%$ of the time, respectively.

5.2.1.2   Phenomenon

The phenomena that we have observed in the data are almost always centered around the simulation of the trajectory of the satellite. The physical interpretations of the phenomena are, not surprisingly, closely related to the topics covered in the couse notes and the topics questioned in the weekly homework. These interpretations range from the crude (e.g., a circular orbit) to the sophisticated (e.g. a centripetal force and its properties). This variety of phenomena show up $100\,\%$ of the time.

5.2.1.3 Comparison between

The comparisons that we have observed in the data are quite varied. Given that the comparison is between the models and phenomena, we expect this characteristic to be the most varied.

5.2.2 Creating data

5.2.2.1 Data set

5.2.2.2 Algorithmic procedure

5.2.2.3 Advance understanding

Chapter 6

Discussion

6.1 Primary practices

Analyzing data, assessing computational models, programming, and thinking in levels are the most common practices that we observed.

6.2 Secondary practices

Creating data, constructing computational models, creating computational abstractions, and understanding the realations within a system are the lesser common practices that we observed.

6.3 Absent practices

Some of the practices that were not observed in the data were collecting data and choosing effective computational tools.

Chapter 7

Conclusion

The most important things that you should be paying attention to are the creation and analysis of data, the assessment and construction of models, programming and troubleshooting, and thinking in levels and communicating

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

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

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

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

[6] Marcos Caballero and Steven Pollock. A model for incorporating computation without changing the course: An example from middle-division classical mechanics. *Americal Journal of Physics*, 2014.

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

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

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

[10] 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.

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

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

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

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

[15] 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.

[16] 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.

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

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

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

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

[21] 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.

[22] 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.

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

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