TABLE OF CONTENTS

Chapter 1

Introduction

Computers continue to revolutionize the way that we do and teach science.

1.1 Computation is important

Computation is important as both a research tool and a pedagogical tool.

1.1.1 As a research tool

Computation is increasingly being referred to as the third leg of modern physics, along with experiment and theory. Its utility in solving complicated problems (e.g., predicting chaotic or non-linear motion) that cannot currently be solved analytically makes it indispensable in many field, from physics to chemistry to biology to engineering.

1.1.2 As a pedagogical tool

Sherwood developed and maintained VPython until it turned in Glowscript. VPython was expressly designed to engage students with computational modeling and the three-dimensional visualization that are characteristic to it.

Chabay et al. have studied computation involving VPython in introductory physics classrooms from a number of different lenses. These lenses range from investigating dif-

ficulties that students run into while computationally modeling, to investigating how to be incorporate computation into the classroom, to investigating how students are thinking computationally.

Although much work has been done around computation at the introductory level, there are still many unanswered or only partially answered questions.

1.2 Computational thinking

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. Even within the fields of education and computer science, many different viewpoints exist on the topic, and the corresponding definitions are just as varied [9]. However, many of these definitions share one fundamental characteristic: solving complex problems through abstraction and analytic thinking with the aid of computer algorithms.

Seymour Papert first introduced computational thinking in terms of students actively constructing knowledge (constructionism) through the production of an artifact (i.e., a computer program). However, Papert does not initially attempt to define computational thinking. Rather, he comments that attempts to integrate computational thinking into everyday life have failed because of the insufficient definition of computational thinking. He optimistically claims that more attempts to define computational thinking will be made and eventually the "pieces will come together [14]." Papert would later go on to say that computational thinking involves "forging new ideas" that are both "accessible and powerful [15]."

Building on Papert, Jeanette Wing defines computational thinking in terms of taking advantage of the processing power of modern computers with the addition of human cre-

ativity. This echoes the core sentiments expressed by Papert: 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 [18].

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 (modeling) is at the heart of computational thinking, and be 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 computer science, new domains of investigation (e.g., introductory physics) require definitions of their own.

Most recently, 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 [13]. 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 physics education. 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 [19]."

One such domain of investigation that could benefit from a clear definition of compu-

tational thinking is that of introductory physics. Many realistic physics problems require thinking abstractly and the computational power of computers (i.e., non-linear forces in Newton's second law). This combination of abstract and algorithmic thinking (computational thinking) is the "heart" of the "computational physics approach" to solving problems [1]. This approach relies heavily on computational thinking as it relates to a particular tool and on both technical and physics computational skills. The AAPT defines 3 technical skills and 7 physics skills. **DC: I think that you can expand on this in Ch 2. Include the table and discuss it.**

Many of the physics skills defined by the AAPT involve modeling and the modeling process. Computational modeling is a powerful problem solving tool. Research shows that computational modeling can be successfully incorporated into the (middle-divison) physics classroom [5]. There are a number of common mistakes students make when solving a Newtonian gravitational problem. We should encourage students to synthesize both analytic and computational skills [6]. **DC: I think you can expand on the prior work that studies computation in physics courses in Ch 2. There's also a number of references that you can use to describe the integration of computation into physics courses. They should appear here.**

Other research on (high school) computational modeling has focused on success. Students' ability to adapt to novel problems seems to rest on the ability to synthesize physics and computational skills [3]. The ability of students to understand the iterative process, that is characteristic of computation, plays a crucial role in the ability to construct novel computational force models.

Therefore, it is important to encourage computational thinking in introductory physics courses. **DC: I'm not sure how this follows. I want a little more of making an**

argument. You are describing what folks have done or said, but I want you to have a point from each of these that supports this statement.

1.3 Computational physics practices

The efficient construction of domain specific knowledge by students has always been one of the ultimate goals of physics education research. The fundamental concepts, ideas, and theories of introductory physics are the foundation for all inquiry in not just the field of physics, but in many related disciplines (e.g., chemistry and engineering). However, there is more to being a productive member of the scientific community than just amassing a collection of facts – it is important that this knowledge be applied in some practical way (e.g., utilizing Newton's second law to numerically predict the trajectory of a rocket). For this reason, the NGSS has broadly defined scientific practices as a combination of both knowledge and skill "to emphasize that engaging in scientific investigation requires not only skill but also knowledge that is specific to each practice [16]." **DC: This is a good paragraph**

Given the recent interest in scientific practices, and computational thinking more specifically, a taxonomy of the computational practices indicative of computational thinking has been defined [17]. 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 so as to be applicable to a wide range of science classrooms.

However, the broad 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. **DC: I think this is where you want to say, and this is what this thesis is about: In Ch 2 we blah blah, and in 3 we blah blah blah, etc.**

Chapter 2

Background

2.1 Historical and Recent comp. research

2.2 Methods for my research

2.3 Framework

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 pre-class homework and a selection of the in-class problems.

3.1 Pre-class tasks

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 achieve this is by requiring students to engage with the fundamental programming ideas before coming to class through pre-class homework and notes. These notes and homework questions highlight the fundamental physical and programming ideas that will be used in class.

For example, consider the portion of the course notes shown in Fig. . These notes are

made available to the students at the beginning of the semester. The content is meant to provide students with a basic understanding of the utility of VPython and a list of common errors.

Figure 3.1: Page of on-line notes introducing computional physics. The picture is just a placeholder.

Similarly, consider the pre-class homework question shown in Fig. that is delivered at the beginning of the third week. 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.), we expect students to be able to draw on and take advantage of this knowledge when faced with a related albeit more complicated problem.

Figure 3.2: Pre-class homework question focusing on the different ways that the magnitude of a vector can be constructed in VPython code.

3.2 In-class tasks

Figure 3.3: A schedule for the semester. The images is just a placeholder.

There are a number of in-class computational problems spread out throughout the semester (see Fig.). The first few computational problems focus on different force models 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. and 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 the associated learning goals[10] were further used to generate the type of in-class project that this study focuses on.

This study focuses on the third and most complicated computational problem delivered to the students, shown in both Figs. and . In order to focus our analysis on specific challenges students demonstrate while working this problem, we developed a framework for analyzing this problem by performing an expert task analysis on the problem. A task analysis consists of breaking a complex "task" (e.g., modeling a gravitational 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 [**?**].

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 were sufficiently described to be useful in video analysis. At the same time, targeted pre-class homework problems were developed to help scaffold students overcoming what we perceived to be challenges based on the task analysis. In doing so, we attempted to place (most) students in the Zone of Proximal Development (ZPD) [**?**]. For example, the pre-class homework problems shown in Fig. **??** were developed to facilitate student understanding of the unit vector of a separation vector between two objects prior to working the computational problem.

Table 3.1: 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.

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

With an agreed upon task analysis, we further focused our video analysis on the sub-tasks that were closely related to the construction of the position dependent Newtonian gravitational force (see Table. 3.1). In order to identify when groups were actually engaging in these sub-tasks, we analyzed groups on two levels: their *speech*, including any and all words or utterances, and the associated *visuals*, including any drawings, writings, lines of code, gestures, or simulation visualizations. Theses two levels were then used to better understand how students overcome the challenges associated with the problem.

Figure 3.4: In-class problem focusing on a non-constant net force model. The picture is just a placeholder at this point.

Chapter 4

Motivation for identifying broader practices

4.1 Debugging

4.1.1 Introduction

Since the development of reasonably affordable and fast computers, educators have argued for their inclusion in the classroom as both a learning aid and a tool.[?, ?] With the recent development of high-level programming languages capable of quickly rendering three-dimensional real-world simulations, the argument for the inclusion of computers in the classroom has not only persisted, but has grown.[8] Well into the 21st century, with its prevalence in modern physics research, we see computation being increasingly referred to as the "third pillar" of physics – along with the more canonical pillars of theoretical and experimental physics.[7]

One physics curriculum that includes a computational element is Matter & Interactions (M&I). The M&I curriculum differs from a traditional one not only through the inclusion of computation (VPython), but also through its emphasis on fundamental physics principles and the addition of a microscopic view of matter.[?, ?] Recent work[11, ?] involving students' use of VPython with the M&I curriculum has begun to analyze the patterns seen in implementing and assessing the use of computation in introductory physics courses.

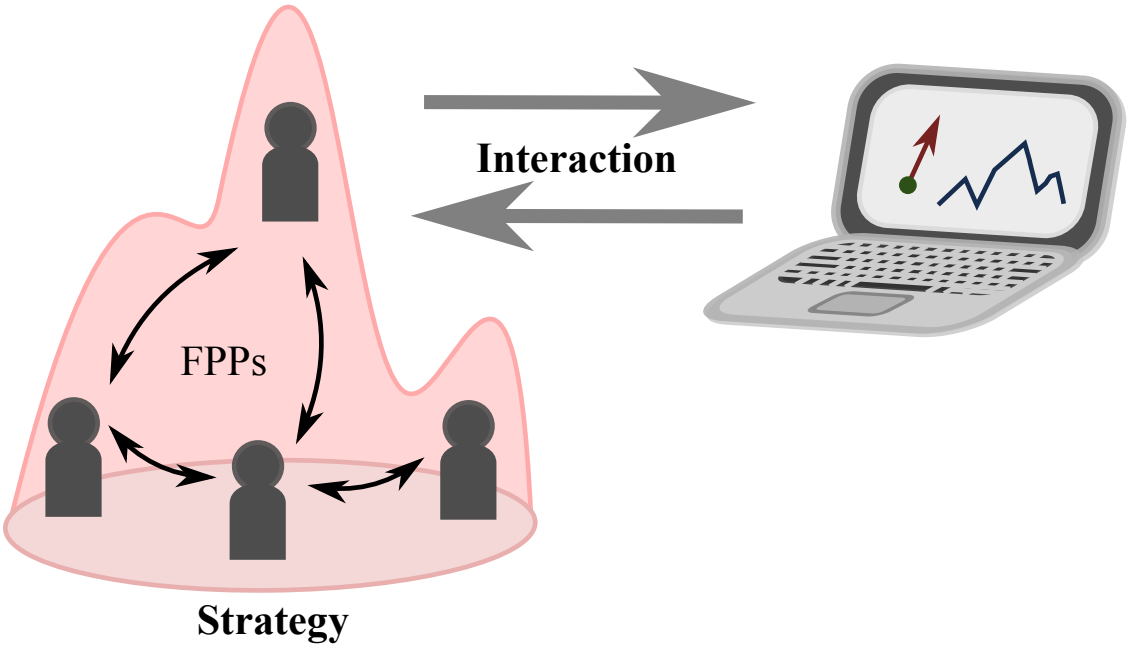
However, numerous questions remain unanswered regarding the processes observed while groups of students work together to model real world phenomena computationally. We extend our research to a novel implementation of M&I with an emphasis on computation in a group setting, called Projects and Practices in Physics (P^3), where students negotiate meaning in small groups, develop a shared vision for their group's approach, and employ science practices to navigate complex physics problems successfully. Borrowing from the literature of computer science education research,[12] we use the notion of computational debugging in a physics context to help uncover the salient practices unique to computational physics problems.

In this paper, 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.

4.1.2 Assumptions

In an increasingly technological world, we have at our disposal computers well suited for the most procedural and tedious, yet indispensable of STEM tasks. Accordingly, in P^3, we treat the computer as an indispensable *modern tool* with which students must familiarize themselves. In spite of the fact that modern computers *take* meaningful direction quite well, they do not yet posses the faculty to *generate* meaningful direction on their own. This necessitates a human factor, in which groups of students must leverage their understanding of fundamental physical principles to model real world phenomena computationally (i.e., generate meaningful direction to a computer).

Figure 4.1: A group of students interacting with VPython where social exchanges focus on FPPs and desirable strategies are being exhibited.



We focus our research 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 strategies (e.g., divide-and-conquer), as illustrated in Fig. 4.1. These group-computer interactions vary in form, from the more interactional process of actively sifting through lines of code, to the less interactional process of observing a three-dimensional visual display – in this work, the more the student interacts with lines of code, the more interactional.

One previously defined computational interaction that reinforces desirable strategies,[**?**] borrowing from computer science education research, is the process of debugging. 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."[12] 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.

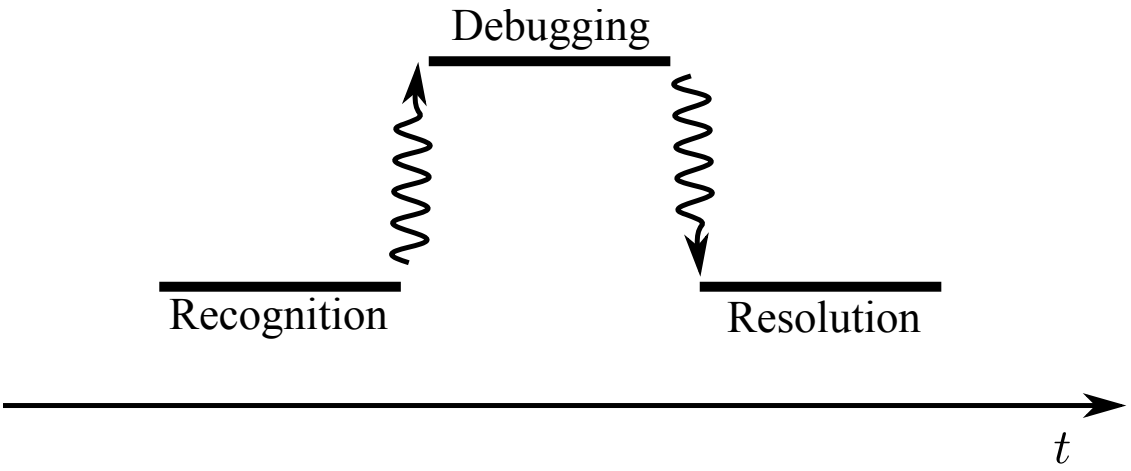
4.1.3 Data

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*.[**?**]

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

This study focuses on one group in the fourth week of class (the fourth computational problem seen) consisting of four individuals: Students A, B, C, and D. The group had primary interaction with one assigned instructor. Broadly, we see a 50/50 split on gender, with one ESL international student. Student A had the most programming experience out

Figure 4.2: The debugging process necessarily corresponds to a phase beset on either side by the phases of recognition and resolution. Note the absence of a vertical scale, as the vertical separation merely acts to distinguish phases.



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

4.1.4 Analysis

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

4.1.4.1 Bug recognition

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

```
SB:  We're stuck.
SD:  Yeah...
```

The simulation clearly displayed the trajectory of the satellite falling into the Earth – not the geosynchronous orbit they expected. This impasse was matched with an indication that they believed the FPPs necessary to model this real world phenomenon were incorporated successfully into the code:

```
SB:  And it's gonna be something really
     dumb too.
SA:  That's the thing like, I don't think
     it's a problem with our understanding
     of physics, it's a problem with our
     understanding of Python.
```

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

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

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

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

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

4.1.4.2 Physics debugging

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

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

In this way, the group not only determined the correctness of lines of code that have been added/modified, but also examined the relationships *between* those lines.

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

SA: Final momentum equals initial momentum
plus net forces times delta t. True?
SC: Yeah...
SB: Yes.
SA: O.K. That's exactly what we have
here. So this is not the problem.

```
     This is right.
SD:  Yeah.
```

That is, Student A *(i)* read aloud and wrote down the line of code $\vec{p}_f = \vec{p}_i + \vec{F}_{\text{net}}\Delta t$ while the entire group confirmed on its correct form. This written line was then boxed, and was shortly followed up *(ii)* with a similar confirmation of the line $\vec{r}_f = \vec{r}_i + \vec{v}\Delta t$ that immediately prompted *(iii)* the confirmation of $\vec{v} = \vec{p}/m$. Thus, not only do we see the group determining the correctness of added/modified lines of code as in *(i)*–*(iii)*, we further see confirmation with the links *between* those lines, as in the velocity from *(ii)* being updated through the momentum from *(i)* given their relationship from *(iii)*.

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

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

Suddenly, a haphazard change to the code:

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

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

4.1.4.3 Bug resolution

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

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

SD:  Is it working?
```

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

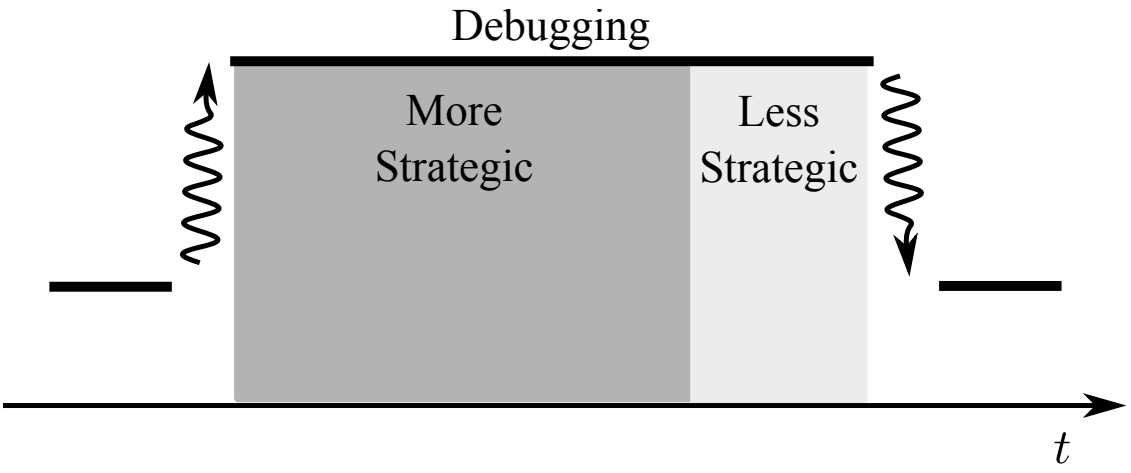
```
SD:  I think that is the issue is that
     we don't have the initial momentum...
SA:  ...momentum correct?
```

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

4.1.5 Discussion

To summarize, in analyzing this particular group, we first identified the two phases in time when the group had recognized and resolved (see Fig. 4.2) a physics bug. We then necessarily identified the phase in-between as the process of physics debugging in P^3, where the fundamentally correct code was taken from producing unexpected output to producing expected output. Given our assumption that the process of computer science debugging encourages desirable strategies, we then began to analyze this process of physics debugging further for strategies unique to P^3.

Figure 4.3: Physics debugging phase consisting of more and less strategic strategies specific to our case study. Note the absence of a vertical scale, as the vertical separation merely acts to distinguish phases.



Given the actions exhibited during the debugging phase, we can separate them into two distinct parts: a more strategic part and a less strategic part, as shown in Fig. 4.3. The group initially gave indication that they were working in a considerate, thorough, and consistent manner, which we classify as more strategic. This is contrasted by the later indications of more haphazard actions, which we classify as less strategic. These are the two physics debugging strategies that, together, led to the resolution of the bug in this context.

The more strategic strategy was exhibited through the confirmation of individual FPPs as well as their relation to others. Not only did the group confirm through discussion, they simultaneously wrote, boxed, and referenced equations in the code – this helped to reduce the number of FPPs they needed to cognitively juggle at any given time.[**?**] This confirmation of FPPs through discussion presented a great learning opportunity for the entire group, where creative and conceptual differences could be jointly ironed-out. Accordingly, we tentatively refer to this strategy as **self-consistency**.

Although the resolution of the bug might not be tied directly to this self-consistency, that does not negate the learning opportunities afforded to the group along the way. Specif-

ically, we saw the group double-checking every fundamental idea used and, possibly more importantly, the links between those ideas. Being physically self-consistent in this manner is a desired strategy in P^3.

The less strategic strategy was exhibited during the haphazard changes to the initial momentum. These changes to the code that eventually resolved the bug, though one of the benefits of computation (i.e., the immediacy of feedback coupled with the undo function), could have been more thoughtful. A deeper understanding of the physics or computation could have tipped the group off to the fact that the initial momentum was too small. Again, this does not negate the learning opportunities afforded to the group through this less strategic strategy, which resembles that of "productive messing about."[**?**] Accordingly, we tentatively refer to this strategy as **play**.

Both of these strategies identified here, self-consistency and play, hold implications for the learning opportunities afforded to groups. More research is needed to dissect these learning opportunities and to deepen our understanding of the strategies themselves.

4.1.6 Conclusion

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

in physics courses.

Chapter 5

Analysis

5.1 Specific methods

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.

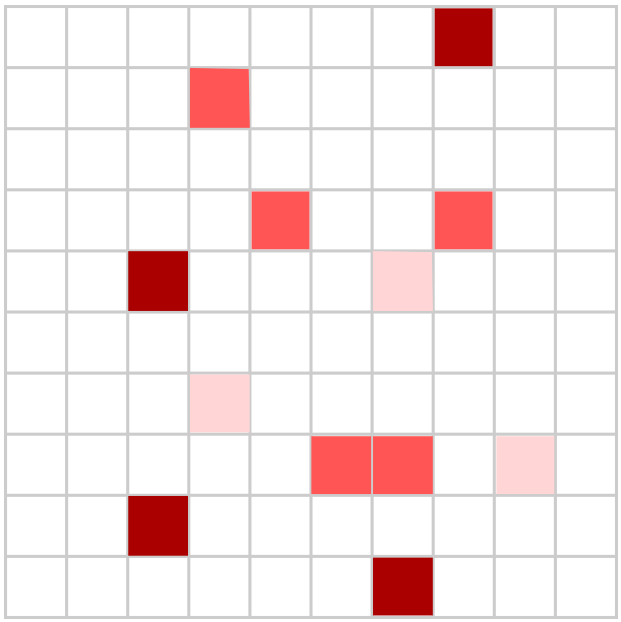


Figure 5.1: A "density plot" for all computational practices and all groups.

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 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 5.2: A table showing rough statistics for assessing computational models.

5.2.1.1 Models

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.

5.2.1.1.1 Group A (gold star)

5.2.1.1.2 Group B (near miss)

5.2.1.2 Phenomena

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.

5.2.1.2.1 Group C

5.2.1.2.2 Group D

5.2.1.3 Comparisons

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.

5.2.1.3.1 Group E

5.2.1.3.2 Group F

Chapter 6

Discussion

Chapter 7

Concluding Remarks

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] John Aiken, Marcos Caballero, Scott Douglas, John Burk, Erin Scanlon, Brian Thoms, and Michael Schatz. Understanding student computational thinking with computational modeling. In *PERC Proceedings*, 2012.

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

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

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

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

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

[12] 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.

[13] 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.

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

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

[16] NGSS Lead States. *Next generation science standards: for states, by states*. The national Academies Press, 2013.

[17] 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.

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

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