Exploring Students' Understanding of Concurrency: A Phenomenographic Study

Filip Strömbäck, Linda Mannila and Mariam Kamkar

Strömbäck, F., Mannila, L., Kamkar, M., (2020), Exploring Students' Understanding of Concurrency: A Phenomenographic Study, *Proceedings of SIGCSE '20*. https://doi.org/10.1145/3328778.3366856

Exploring Students' Understanding of Concurrency – A Phenomenographic Study

Filip Strömbäck
Department of Computer and Information Science
Linköping University
Linköping, Sweden
filip.stromback@liu.se

Linda Mannila
Department of Computer and Information Science
Linköping University
Linköping, Sweden
linda.mannila@liu.se

Mariam Kamkar
Department of Computer and Information Science
Linköping University
Linköping, Sweden
mariam.kamkar@liu.se

ABSTRACT

This paper continues previous efforts in understanding the problems students face when learning concurrency. In this paper, we explore students' understanding of the subject using phenomenography in order to gain insights that can aid in explaining the underlying causes for common student mistakes in concurrency, which has been studied in depth previously. Students' experience of concurrency and critical sections were analyzed using a phenomenographic study based on interviews with students attending one of two courses on concurrency and operating systems. We present 6 categories describing students' experience of concurrency, and 4 categories describing students' experience of critical sections in this paper. Furthermore, these categories are related to previous results, both to explore how misconceptions in the categores relate to student mistakes and to estimate how common it is for each category to be discerned.

CCS CONCEPTS

• **Applied computing** → **Education**; • **General and reference** → *Empirical studies*; • **Theory of computation** → *Concurrency*.

KEYWORDS

computer science education, concurrency, critical sections, phenomenography

ACM Reference Format:
Filip Strömbäck, Linda Mannila, and Mariam Kamkar. 2020. Exploring Students' Understanding of Concurrency – A Phenomenographic Study. In *The 51st ACM Technical Symposium on Computer Science Education (SIGCSE'20), March 11–14, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3328778.3366856

1 INTRODUCTION

Concurrency is often percieved as a difficult subject by students. This is clearly shown in a study by Strömbäck et al. [16], where the authors find that less than half of students failed to synchronize an implementation of a simple data structure. The study also provides insights into what mistakes were common and attempt to draw conclusions regarding what underlying misconceptions could cause the mistakes. These conclusions were, however, vague due to the lack of qualitative data.

Therefore, this paper aims to provide additional qualitative data in order to gain further insights into students' understanding of *concurrency* and *critical sections* in order to better understand why certain kind of mistakes are common. These insights can then be used to suggest parts of the subject that should be paid additional attention when teaching in order to avoid the misconceptions and in turn also the common mistakes. Specifically, we aim to answer the following research questions:

(1) In what ways do students experience *concurrency* in general and *critical sections*?
(2) How do these understandings relate to common mistakes when synchronizing code?

This is similar to previous work by Lönnberg and Berglund [11], who explore students' understanding of *tuple spaces*, which is a synchronization mechanism that highlights the need for coordination between tasks [3, 8]. This paper, however, focuses on concurrency in general, which is not explicitly addressed by Lönnberg and Berglund.

In this paper, we will first introduce concurrency, teaching concurrency and phenomenography in Section 2, followed by a description of the method used to collect and analyze data in Section 3. Section 4 presents the results from the phenomenographic analysis and Section 5 relates the results to previous work and explores the implications for teaching, followed by a conclusion in Section 6.

2 BACKGROUND

In this section, we briefly introduce the aspects of concurrency relevant to this paper, related work in teaching concurrency, as well as phenomenography, which is used to examine students' understanding of the subject.

2.1 Concurrency

Modern operating systems typically provide two concepts used for concurrency: *processes* and *threads*. A process describes an instance of a running program. Processes are generally isolated from each other (e.g. no shared memory), but may communicate using mechanisms provided by the operating system (e.g. the file system). Each process contains one or more *threads*, each representing a sequence

SIGCSE '20, March 11–14, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6793-6/20/03…$15.00
https://doi.org/10.1145/3328778.3366856

of instructions that may execute concurrently. All threads in a process share resources associated with the process. In this paper, we will use the term *thread* to describe entities executing concurrently, as the differences between processes and threads are not always apparent when working in the operating system's kernel, as students do in the examined courses.

There are two major ways to achieve concurrent execution on a particular platform, and they are often used together. First and foremost, the hardware may provide true parallelism in the form of multiple cores or multiple CPUs executing multiple instructions in parallel. Additionally, preemptive or non-preemptive multitasking implemented in software may be used to alternate between multiple threads (usually quickly) to create the illusion of parallel execution.

The second concept explored in this paper is *critical sections*. A critical section is a part of a concurrent program that would cause undesired or unpredictable behaviour if executed concurrently. This is typically the case when a shared resource (e.g. a shared variable) is used. In order to avoid the problems associated with concurrent execution, critical sections need to be protected in some way, usually with a lock. When protecting a critical section, one effectively reduces the amount of code that may execute concurrently. Therefore, it is important to understand the extends of the critical section as to not overly reduce the concurrency of the program, thereby reducing the benefits of using concurrency.

2.2 Teaching Concurrency

Moström [14] notes in a literature study that much work has been done in the area of teaching concurrency. A large number of visualizations, languages, and other tools [3, 15] have been created to aid teaching. However, Moström notes that much of the work do not evaluate the benefits of the proposed tools on student learning in sufficient depth, leading to the conclusion that not much is known about teaching concurrency.

One area highlighted as promising by Moström is studying programming errors made by students, which is done by Lönnberg, Kolikant and Strömbäck et al. among others. Lönnberg [9] explores misunderstandings held by students by examining students' solutions to three concurrency assignments in a course, and finds that a large portion of errors in the assignments can be attributed to misunderstanding either some part of the task itself, synchronization goals of the task or the computational model used. Lönnberg et al. [10] further investigate the reason behind the misunderstandings through a phenomenographic study regarding one of the tasks in the course, and find that some students experience the task as a requirement to get a grade, or as an idealized problem that will not work in practice anyway. This way of approaching the assignments could explain at least some of the mistakes. The study also covers how students test concurrent programs, where the authors find instances ranging from unplanned testing to some degree of correctness proofs.

Students' mistakes are also examined by Strömbäck et al. [16], who examine 216 student solutions to a synchronization assignment. The assignment consists of multiple parts and involves identifying and eliminating busy-wait as well as highlighting and synchronizing critical sections in a piece of code. Since the assignment asks students to highlight parts of their reasoning, and to synchronize a data structure that could be instantiated multiple times, the collected data illustrate many important aspects of synchronization. Contrary to the previously mentioned studies by Lönnberg et al., however, the focus is on the mistakes themselves rather than the underlying misunderstandings. The authors present possible causes for the mistakes, but are unable to draw definitive conclusions due to lack of qualitative data.

Similar ideas are also explored by Kolikant [7], who examined high school students' views of correctness using questionnaires and found that students' views often differ from the teachers's views. This explains why students find some solutions satisfactory while the teacher does not. In another study [5], Kolikant explores another cause of errors found by Lönnberg: misunderstanding the computation model. Kolikant observed students' mistakes during laboratory sessions and interviewed one of the students. From this, the author notes that some students initially experience the concurrent system as a user, without trying to understand what happens inside the system, and that students need to change their perspective to properly understand the models and solve problems. A change in their perspective was also observed by Ben-Ari and Kolikant [1] in a study of an introductory course on concurrency for high school students. In the course, some students pointed out that in contrast to sequential programming, which could be solved by trial and error, concurrency requires a higher degree of understanding and thinking to solve problems correctly. This, along with further observations from the same study [6], highlights another necessary change in their perspective similar to the previous one.

2.3 Phenomenography

Phenomenography is a qualitative research technique to explore students' perception and experience of a particular object of learning [2]. As such, it is a second-order perspective; we are not interested in the phenomenon itself, but the students' experience of the object. According to phenomenography, learning is the process of successively discovering new aspects of an object of learning, where each aspect represents a *degree of variation*, i.e. an aspect which differs when compared to other objects [12, 13]. Thus, a phenomenographic study aims to find *variation* in the descriptions of an object of learning in order to identify the relevant aspects experienced by a learner.

Data for a phenomenographic study is typically collected through semi-structured interviews. A researcher prepares a number of questions asking the subject to describe their experience of a particular phenomenon. Follow-up questions are then asked as appropriate in order to fully explore how the subject understands the object being studied. The interviews are then transcribed and the researcher proceeds to read and re-read the transcripts, first in order to extract quotes that highlight some aspect of the object of learning, and then to categorize the extracted quotes into a fairly small number of categories, each describing some aspect of the object of learning, or a combination of aspects in other categories that allow new insights. This process can be viewed as a kind of sorting where the final categories are initially unknown, but are discovered and iteratively refined during the sorting process [2].

The number of participants in a phenomenographic study is usually low compared to other kind of techniques. Studies involving

fewer than ten participants are not uncommon [4, 11], which could raise concerns regarding the validity of the results. Due to the low number of participants, it is indeed problematic to draw quantitative conclusions from the data (e.g. which is the most common way of understanding something). The goal of phenomenography is, however, to find and summarize the different ways in which a phenomenon is experienced in a group of students as a whole [2]. Even though the results may originate from a relatively small group, Berglund [2] and Marton [12] describe multiple instances where the results have been applied to teaching with good results.

3 METHOD

Data was collected by interviewing students taking one of two similar bachelor level courses on *concurrency and operating systems* at Linköping University. Both courses contain laboratory assignments where students are asked to implement different functionality (e.g. system calls, process synchronization) in the educational operating system Pintos[1]. The main difference between the courses is that one assumes previous theoretical knowledge on operating systems, which is taught in a separate course, while the other does not. Both courses do, however, teach concurrency and synchronization as those particular concepts are not taught in a previous course. Additionally, the first course is given for students attending a 3 year computer science program, while the second is given for students attending a 5 year program.

Students from the first course, taught by the first author, were invited during one of the final lectures and by e-mail, while students from the second course were only invited by e-mail as the final lecture of the course had already been held. In order to encourage students to participate, students were offered coffee and cake during the interview. Furthermore, to avoid influencing students' answers by the student-teacher relationship during a course, interviews in the first course before the final exam were conducted by a colleague not related to any of the courses.

The interviews lasted between 30 and 70 minutes, during which the interviewer asked questions regarding the students' understanding of different topics related to concurrency, the two relevant to this paper are: "How would you describe concurrency?" and "How would you describe a critical section?". In order to properly explore the student's understanding of the topic, follow-up questions were asked as needed. A number of such follow-up questions were planned in advance, but additional questions were asked as needed in order to properly explore or clarify previous answers. Since this study concerns how students experience the topics rather than if their understanding is correct, any incorrect answers during the interviews were explored in equal depth to correct answers. The interviews were recorded, transcribed, and finally analyzed by selecting quotes from the transcripts and categorizing them according to the method described in Section 2.3. If a quote contains a reply to a question other than one of the two main question, the relevant follow-up question is included in the quote.

A total of 14 students participated in the study, 5 from the first course and 9 from the second. All of the participants were male, most likely since the majority of students attending the courses are male.

[1] https://web.stanford.edu/class/cs140/projects/pintos/pintos.html

4 RESULTS

In this section we present the results from the phenomegraphic analysis. We present the different ways in which the concept *concurrency* as a whole is understood in Section 4.1, the different ways in which *critical sections* are understood in Section 4.2.

The different categories are illustrated with excerpts from the interviews. The interviewers are labeled **I** and **J**, the students from the first course are labeled **M** to **Q** and the second course **R** to **Z** in order to preserve their anonymity. Most interviews were conducted in Swedish. The interviews were analyzed in their original language and the selected excerpts were translated into English.

4.1 Concurrency

Several different ways of experiencing *concurrency* were discerned by the students. As can be seen in Figure 1, these can be categorized into three basic aspects. As indicated by the arrows, these insights can then be combined in different ways to draw higher level conclusions about the behavior of the final program, such as *performance* (4.1.4) and *responsiveness* (4.1.6).

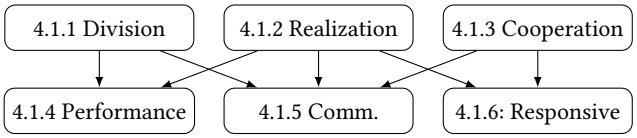

Figure 1: Summary of the students' views of *concurrency*

4.1.1 Division of a problem. This category experiences concurrency in terms of how sub-problems should be structured to allow solving them concurrently. This is illustrated by student **M**, who describes concurrency as the process of dividing a problem into smaller parts that can execute concurrently:

> **M:** I would describe it [concurrency] as a way of working... simultaneously at a problem. A problem that you are able to divide into two or more pieces that can be worked on simultaneously.

From this excerpt we can see that the student is aware of (at least) two kinds of problems: problems that are divisible into parts that can execute concurrently, and problems that are not divisible in this manner. Therefore, the variation in this category is *the ability to decompose problems into independent sub-problems*, which is also shown by student **U**:

> **U:** Um, I would say... actions that can be executed simultaneously and... well, isn't it the concept that you... But is often used for actions that can be executed simultaneously and that you can divide. [...]

4.1.2 Realization. This category experiences concurrency in terms of how concurrency is achieved in a particular system. That is, students realize that concurrency can be achieved in multiple ways, for example by preemption and by exploiting the parallelism provided by the hardware, or a combination thereof. The variation in this category is the different ways of *realizing* concurrency in a system. This is illustrated by student **S**, who experiences concurrency in terms of the two major ways of achieving brought up in the course: preemptive multitasking and true parallelism:

S: I experience it as trying to do multiple things in parallel. It doesn't have to be physically parallel... [...] It can also be that a program takes... I don't remember the proper word... But when you have... when you do one thing, then you pause everything, save your settings in a special field, then you jump to. Similar to how a CPU is doing today, so that you may have multiple threads.

4.1.3 Cooperation. In this category, focus lies on the complexity associated with running multiple, largely unrelated, threads simultaneously, and how using concurrency impacts the implementation of such programs. This is best illustrated by student **Z**s explanation:

I: In what way? What are we trying to achieve?
Z: Well... We want to keep multiple balls in the air, stated clumsily. But, it becomes very bothersome to in every program you develop think that everything need to happen in one sequence without interruptions here and there. For example, we want to be able to do user input.

Here, we can see that the student realizes the difficulties associated with implementing a single thread that waits for input from the user while doing something else in the background, and sees variation in the two solutions to the same problem. Without concurrency, the single thread would need to keep track of both things simultaneously and perhaps abort the background task whenever input from the user arrives. With concurrency, however, this task is delegated to the scheduler and the two threads can work independently of each other without taking the other into account. This is what is meant by *cooperation* in this case, but it could also be seen as *separation* of independent tasks.

4.1.4 Performance. This category combines the aspects from the categories *division of a problem* (4.1.1) and *realization* (4.1.2). Additionally, the aspect of the resulting program's performance, and how it is affected by the problem division and realization of concurrency is experienced. This is illustrated by student **M**, who states that the main goal of concurrency is improved performance, and relates the performance to the hardware parallelism and the ability to divide problems into independent parts:

J: What would you say is the greatest benefit of using concurrency?
M: Um... The biggest difference is... I guess the biggest thing is to be able to squeeze a bit more performance, to be able to do a bit more simultaneously. There is a limit to the clock frequency of a CPU, so you need to do many things simultaneously. But there is a bit... there are dependencies between... things are done in a certain order for a reason most times, so it is not certain that you are able to do that much simultaneously anyway then.

4.1.5 Communication. This category focuses on how the interactions between *division of a problem* (4.1.1), *realization* (4.1.2), *cooperation* (4.1.3) and the need for *communication* or *synchronization* between different threads. This interaction is expressed by student **Y** when asked about concurrency:

Y: Yeah... For me, concurrency is like... separating things, and be sure that everything works fine, even with this separation. If I... I've been like working with matrices during my internship last year, and basically it was huge matrices, so I had to separate data to allow faster computation, and... Well the thing was sharing memory is good in a way, and not good in another way. Because if some people are trying, well two cores are trying to read and write at the same place, for example, you can have huge problems. So, you have to schedule to separate to wait or not to wait, and to... Yeah, basically that is what I understand when someone says concurrency.

Here, the student makes the connection between the properties of the environment the student is working in and the need for synchronization to make the final program work properly.

4.1.6 Responsiveness. This category focuses on the interaction between *realization* (4.1.2), *cooperation* (4.1.3) and the responsiveness of a program, i.e. how quickly a program responds to some external or internal event. This is illustrated by student **X**:

I: Is there any particular property of your program that is important to consider in order to make it efficient when using threads?
X: The first thing that comes to mind is to do things in the right order... To keep track of when things are done... I have one experience of threading, the project we did recently, called "Microcomputer Project". Then we opted to make two threads in our robot. One in charge of communication, and one in charge of most of the movements and thinking. I did not work on it myself, but my colleagues described it as... since they were working in parallel, they can make... there is less time stalling. Less time where you are unable to do something because you need to wait for something.

Here, the student describes using threads to execute different tasks to avoid unnecessary waiting, just as is the case in the *cooperation* category. However, the student also connects this with insights from the *realization* category to conclude that these two aspects combined allow the program to respond quicker to events because the system is able to execute the thread responsible for the event immediately, rather than waiting for a single thread to take care of the event whenever it has time.

4.2 Critical Sections

As shown in Figure 2, two main aspects of critical sections were discerned by the students: *code* (4.2.1) and *data* (4.2.2 and 4.2.3), which associate a critical section with the associated code or data respectively. As indicated by the arrows, these two aspects are then combined in the final category *atomic* (4.2.4), where critical sections are experienced as a combination of the two: as an operation that is executed atomically in relation to some other operations.

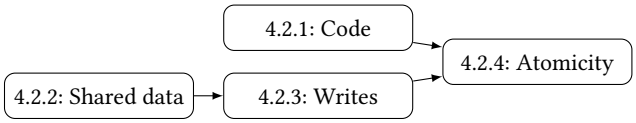

Figure 2: Summary of the students' views of *critical sections*

4.2.1 Code. The focus of this category is how the *code* inside a critical section is executed, which is illustrated by the student **Q**:

Q: Well, it is like we heard in the course. It's a place where no one else may disturb, where no context switches may occur. You are changing something, and you should not be disturbed, really.

Student **Q** describes a critical section as a place where no *context switches* may occur, implying that the current thread should not be interrupted by other threads when executing inside a critical section. This is in contrast to other code, outside critical sections, where execution of a thread may be interleaved with other threads at any time. Note that this observation, as well as other in this category, do *not* consider the data being manipulated, only the code itself.

4.2.2 Shared data. The focus of this category, in contrast to the previous, is the *data* accessed inside a critical section, which is illustrated by student **R**:

R: A critical section could be the part of the code that is actually shared by multiple threads that is common to all of the threads. That could be some sort of shared memory, or that could be, I remember a few scenarios that it was basically some container, like a vector, and you didn't want to have two threads operating on that at the same time because you might not get the result you were hoping to. [...]
I: And what would you say, like, the critical section consists of? You mentioned some code, some data...
R: Yeah, I suppose the critical section... the actual critical section should be the data itself that you don't want to... that you need to be careful about accessing concurrently. [...]

The student describes a critical section in terms of shared data manipulated concurrently, which could cause unexpected results without proper synchronization. This is seen in contrast to data that is not shared, where these kind of problems do not occur. Even though the student initially mentions code, we can see from the remainder of the quote, especially the second question from the interviewer, that focus is almost exclusively on the data rather than the code.

4.2.3 Writes. This category opens up a new aspect of critical sections together with the one found in the *shared data* category. Namely, the aspect of *how* data is accessed. This aspects makes it possible to realize that reading shared data concurrently is generally not a problem, as long as no thread writes to the shared data. This, in turn, allows more fine-grained reasoning about which operations need synchronization, which is illustrated by student **O**:

O: Hmm... When two processes are in conflict. When at least one... when they share data, and one of them wants to... at least one of them wants to write while the other either wants to read or write. And it's like... that's the definition from our database course. It is how they describe it. [...]

4.2.4 Atomicity. The final category combines the aspects brought into focus in the previous categories, which allows creating the notion of an operation that should be atomic in relation to zero or more other operations on the same data. The connection between code and data allows associating different operations operating on the same data to the same critical section. Additionally, it allows for more fine-grained locking as the connection between code and data allows realizing that executing the same code for different pieces of data is not a problem. This is illustrated by student **Z**, which highlights all of these aspects when describing a critical section:

Z: Well, a critical section as I think of it, is just a section we need some kind of synchronization around. That... Well, it could be that multiple try to read a value or a variable simultaneously... but it could be modified, either there, or from somewhere else, and then we don't want... but we need exactly what was there at the moment we entered the critical section. [...]

5 DISCUSSION

In this section we will examine the results presented in Section 4, and relate them to previous work and observations in computer science education, mainly the previously mentioned study by Strömbäck et al. [16] which examined 216 answers to a problem where students were asked to synchronize a small data structure. Even though the aim differs from this paper, the data is useful both to validate the results in this paper, and to attempt to quantify how common it is to discern the different categories.

5.1 Concurrency

As shown in Figure 1, six ways of experiencing concurrency were discerned by the students. Even though the study by Strömbäck et al. [16] focus on synchronization errors rather than concurrency itself, failure to discern all but one of the aspects found here are potentially visible as mistakes in the study, which shows the relevance of the aspects presented in Section 4.1.

First and foremost, failure to discern the *division of a problem* aspect (4.1.1) would lead to having difficulties in the decomposition of a problem into multiple independent parts. This corresponds well to the issue of using global locks to synchronize multiple data structures rather than separate locks for each instance in the study; failure to see the different independent subproblems in this case makes using global locks a viable solution. The related aspect of *performance* (4.1.4) also affects the locking granularity. Discerning the aspect of how the synchronization relates to performance allows realizing that each element in the array is actually independent and can be synchronized individually rather than requiring one lock for the entire array.

Another interesting observation from the study was that some students used a mental model of concurrency with too strong guarantees compared what is typically provided by languages and hardware. This misconception closely relates to the *realization* aspect (4.1.2), as the misconception clearly stems from an incomplete picture of the subtleties of the realization. For example, assuming that only a single CPU core is used or that caches nor compiler optimizations are present makes it easy to use a model with too strong memory coherency, which is exactly what was found.

Furthermore, the *cooperation* aspect (4.1.3), where the cooperation of multiple threads is in focus, is illustrated very well by part C of the assignment explored by Strömbäck et al. [16], which asked students to show how multiple interacting threads may cause a specific problem. The authors note that a large portion (around 80%) of students provided a correct example, indicating that most students discern this aspect of concurrency. The related *communication* aspect (4.1.5), which additionally focuses on the communication

between threads required to avoid problems, is less common, as can be seen by the number of students who correctly eliminate the concurrency issues present (62% for part A and 44% for parts D and E). The final category, *responsiveness* (4.1.6) is not found in the study, as responsiveness is not explored by the assignment.

Aspect	Discerned by
Cooperation (4.1.3)	$\approx 80\%$
Division (4.1.1)	$\approx 62\%$
Communication (4.1.5)	$\approx 44 - 62\%$
Performance (4.1.4)	$\approx 12\%$

Table 1: Estimation of commonly discerned aspects

Since most of the aspects found in this paper can be related to mistakes in the study by Strömbäck et al. [16], we can use those results to estimate which aspects of concurrency are commonly discerned by students. This will only be a rough estimate since the aspects are not isolated to individual types of mistakes in the study. The estimate may still be useful, however, as teaching effort can be directed towards the aspects that students find most difficult. Table 1 contains the estimation for all aspects except for *realization* (4.1.2) and *responsiveness* (4.1.6). While *realization* is likely discerned to some extent by most students, the realizations possible from these insights depend on which possible realizations are visible to the student. For this reason, it is difficult to quantify this aspect from the available data. Responsiveness is, as previously mentioned, not examined in the study at all and therefore not included.

5.2 Critical Sections

Four different ways of experiencing critical sections were discerned by the students, as shown in Figure 2. In this case, a critical section was either associated with its associated code (4.2.1), its associated data (4.2.2 and 4.2.3) or both (4.2.4). Failure to realize at least one of the aspects found here explain most of the mistakes related to critical sections found by Strömbäck et al. [16].

The most common mistake found in the study (27% of the students) was to synchronize two separate functions manipulating the same data using different locks. The authors hypothesize that this might be due to students only associating critical sections with the code rather than also including the data. This corresponds to a student having discerned the *code* aspect (4.2.1) while failing to discern any of the other aspects. This excludes the 4% of students who motivated why it would be safe to use different locks, since that misunderstanding is more likely caused by assuming that preemptive multitasking is used, which is more accurately captured by the *realization* category (4.1.2) mentioned previously.

Another common mistake (around 25% of the students) was specifying one or two small critical sections rather than one large, only including individual reads or writes to shared data in the critical sections. This shows a focus on the access to shared data, rather than on what the code is actually using the data for. Therefore, both of these cases describe mistakes likely caused by students who have discerned the *shared data* (4.2.2) and possibly also the *writes* (4.2.3) aspects of a critical section, but not the *code* aspect (4.2.1). Since the majority of students who only used one critical section only included the write to shared data while opting to ignore the read, we can conclude that students with a too small critical section likely discerned the *writes* aspect (4.2.3) while the 3% of students who used multiple critical sections likely did not. The remaining category, *atomicity* (4.2.4), represents students who discerned the other three aspects.

Aspect	Discerned by
Code (4.2.1)	$\approx 79\%$
Shared data (4.2.2)	$\approx 73\%$
Writes (4.2.3)	$\approx 70\%$
Atomicity (4.2.4)	$\approx 49\%$

Table 2: Estimation of commonly discerned aspects

Even though phenomenography do not attempt to predict the amount of students who discern the categories, we can use the above observations to draw such conclusions with the help of the data collected by Strömbäck et al. [16]. These conclusions are presented in Table 2. Note that the numbers are approximate since the categories for incorrect critical sections are separated by the two functions, but the amount of overlap is not mentioned. Since it is likely that a student who makes a mistake in one of the functions makes the same mistake in the other, we assume that the overlap is as large as possible. Note, however, that the possibility of the mistakes being caused by a lacking understanding of parameters, references and scope, as mentioned by the authors, can not be excluded by this study. More research is needed to conclude whether the misconceptions originate from a lacking understanding of programming fundamentals or concepts related to concurrency.

6 CONCLUSIONS

In this paper, we explore students' understanding of concurrency and critical sections using phenomenography. Six different aspects of concurrency and four aspects of critical sections were discerned by the students, and are presented in Section 4. Additionally, it is useful to know which aspects are commonly discerned by students so that teachers may focus on the less commonly discerned aspects when teaching. However, this is a question left unanswered by phenomenography due to the small number of participants. As such, we estimated which aspects are commonly discerned in Section 5 by connecting each category to mistakes in a quantitative study by Strömbäck et al. [16]. The results in Tables 1 and 2 show that *communication* and *performance* are aspects that are not often discerned within concurrency, and that *atomicity* of an operation is seldom discerned for critical sections. Furthermore, the results suggest that the *shared data* aspect and the *code* aspect of critical sections are about equally common.

We conclude by summarizing the research questions presented in Section 1:

(1) Students experience concurrency in six different ways, as presented in Section 4.1, and critical sections in four different ways, as presented in Section 4.2.
(2) All aspects discovered are visible as common mistakes when synchronizing code, with the exception of *responsiveness*, as discussed in Section 5.

REFERENCES

[1] Mordechai Ben-Ari and Yifat Ben-David Kolikant. 1999. Thinking Parallel: The Process of Learning Concurrency. In *Proceedings of the 4th Annual SIGCSE/SIGCUE ITiCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '99)*. ACM, 13–16. https://doi.org/10.1145/305786.305831

[2] Anders Berglund. 2006. Phenomenography as a way to research learning in computing. *Bulletin of Applied Computing and Information Technology* 4, 1 (2006). http://www.citrenz.ac.nz/bacit/0401/2006Berglund_Phenomenography.htm

[3] Nicholas Carriero and David Gelernter. 1989. Linda in Context. *Commun. ACM* 32, 4 (April 1989), 444–458. https://doi.org/10.1145/63334.63337

[4] Bernard Doyle and Raymond Lister. 2007. Why Teach Unix?. In *Proceedings of the Ninth Australasian Conference on Computing Education - Volume 66 (ACE '07)*. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 19–25.

[5] Yifat Ben-David Kolikant. 2004. Learning Concurrency as an Entry Point to the Community of Computer Science Practitioners. *Journal of Computers in Mathematics and Science Teaching* 23, 1 (2004), 21–46.

[6] Yifat Ben-David Kolikant. 2004. Learning concurrency: evolution of students' understanding of synchronization. *International Journal of Human-Computer Studies* 60, 2 (2004), 243–268. https://doi.org/10.1016/j.ijhcs.2003.10.005

[7] Yifat Ben-David Kolikant. 2005. Students' Alternative Standards for Correctness. In *Proceedings of the First International Workshop on Computing Education Research (ICER '05)*. ACM, 37–43. https://doi.org/10.1145/1089786.1089790

[8] Sirong Lin and Deborah Tatar. 2011. Encouraging Parallel Thinking Through Explicit Coordination Modeling. In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education (SIGCSE '11)*. ACM, 441–446. https://doi.org/10.1145/1953163.1953292

[9] Jan Lönnberg. 2006. Student Errors in Concurrent Programming Assignments. In *Proceedings of the 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006 (Baltic Sea '06)*. ACM, 145–146. https://doi.org/10.1145/1315803.1315833

[10] Jan Lönnberg, Anders Berglund, and Lauri Malmi. 2009. How Students Develop Concurrent Programs. In *Proceedings of the Eleventh Australasian Conference on Computing Education - Volume 95 (ACE '09)*. Australian Computer Society, Inc., 129–138. http://dl.acm.org/citation.cfm?id=1862712.1862732

[11] Jan Lönnberg and Anders Berglund. 2007. Students' Understandings of Concurrent Programming. In *Proceedings of the Seventh Baltic Sea Conference on Computing Education Research - Volume 88 (Koli Calling '07)*. Australian Computer Society, Inc., 77–86. http://dl.acm.org/citation.cfm?id=2449323.2449332

[12] Ference Marton. 2014. *Necessary Conditions of Learning*. Routledge.

[13] Ference Marton and Shirley Booth. 1997. *Learning and Awareness*. Routledge.

[14] Jan Erik Moström. 2011. Learning concurrency - What's the problem? In *A study of student problems in learning to program*. Umeå, Sweden, Chapter VII. http://urn.kb.se/resolve?urn=urn:nbn:se:umu:diva-48216

[15] Mitchel Resnick. 1990. MultiLogo: A Study of Children and Concurrent Programming. *Interactive Learning Environments* 1, 3 (1990), 153–170. https://doi.org/10.1080/1049482900103O1

[16] Filip Strömbäck, Linda Mannila, Mikael Asplund, and Mariam Kamkar. 2019. A Student's View of Concurrency - A Study of Common Mistakes in Introductory Courses on Concurrency. In *Proceedings of the 2019 ACM Conference on International Computing Education Research (ICER '19)*. ACM, 229–237. https://doi.org/10.1145/3291279.3339415