Evaluation of a structured design methodology for concurrent programming

A. Bijlsma
Open Universiteit
6401 DL Heerlen, The Netherlands
lex.bijlsma@ou.nl

C. Huizing
Eindhoven University of Technology
5600 MB Eindhoven, The Netherlands
c.huizing@tue.nl

R. Kuiper
Eindhoven University of Technology
5600 MB Eindhoven, The Netherlands
r.kuiper@tue.nl

H.J.M. Passier
Open Universiteit
6401 DL Heerlen, The Netherlands
harrie.passier@ou.nl

H.J. Pootjes
Open Universiteit
6401 DL Heerlen, The Netherlands
harold.pootjes@ou.nl

J.E.W. Smetsers
Radboud Universiteit
6500 HC Nijmegen, The Netherlands
s.smetsers@cs.ru.nl

ABSTRACT

Learning how to design and implement a program is hard. Teaching methods and textbooks on Java programming often treat a new subject in terms of syntax and examples. Little attention is paid to systematically designing programs with these new concepts. Research has shown that such a complex task requires not only conceptual knowledge, but also explicit procedural support.

In this paper, we investigate the effect of combining conceptual and procedural guidance to teaching and learning concurrent programming. We build on earlier research in which we have introduced such a structured design methodology which divides the development of multi-threaded Java programs into a sequence of explicit, manageable steps: the Steps Plan.

We present our experiences with applying the Steps Plan in an introductory course on object-oriented programming, with multi-threading. The main questions addressed are: "What problems did the students encounter in direct relation to the Steps Plan?", and "What general problems surfaced?"

As to the first question, two important issues were that using a relatively far developed sequential solution as a stepping stone towards a multi-threaded solution wrong-footed some students, and that remedying race condition situations turned out to be supported at a too high level of abstraction.

As to the second question, two notable issues were that deciding on the right amount and type of concurrency by themselves is maybe too difficult for students at this level, and that the notion of (establishing) correctness or quality of a solution is, different from the sequential case, not intuitively clear to students.

For these issues, the paper recommends remedies and indicates directions for future work. We discuss the consequences for education as regards teaching materials and forms of teaching.

CSERC '19, 18 – 20 November 2019, Larnaca, Cyprus

ACM ISBN 978-1-4503-6338-9/17/11…$15.00
https://doi.org/10.1145/3162087.3162088

CCS CONCEPTS

• **Computing methodologies** → **Concurrent programming languages**.

KEYWORDS

Education, object-oriented programming, concurrency, Java, program design

1 INTRODUCTION

Many students have difficulties in learning to program. Programming is a very complex activity that requires effort and a special approach both in the way it is learned and taught. Traditional programming courses and standard textbooks on programming focus on conceptual knowledge rather than on procedural guidance of students. They mainly rely on the learn-by-example principle: After briefly introducing a new concept or language construct, and illustrating these new notions with examples, students are expected to construct their own programs 'by analogy'. The actual programming skills are then developed in supervised lab classes. This approach, while quite labour-intensive on the teaching side, provides the student with no methodical procedure for starting and systematically navigating the complex programming task. The lack of procedural guidance is often felt as ineffective. Kirschner, Sweller and Clark [12] argue that learning such a complex task not only requires conceptual knowledge, but also explicit procedural assistance. Essential is to understand which steps to take in order to solve the problem, as well as how to recognize a potential solution.

In this paper, we present our experiences with applying such a combined approach to teaching and learning concurrent programming. To support this complex task, we proposed a design methodology that provides scaffolding for the development of programs in the form of a sequence of explicit, manageable steps (known as procedural guidance [20]): the Steps Plan, as presented at the CSERC 2017 conference [1]. This paper is part of an educational design research project [16] in which we improve our programming education through cycles of developing a concrete change in our teaching, applying it at several universities, collecting data on it and analyzing the effects.

To set the context, we briefly summarize the Steps Plan – for further detail we refer to our earlier paper [1]. The idea of the Steps Plan is to provide beginning students with a design method for

simple concurrent programs in which the development of a program is divided into explicit, manageable steps. Each step consists of a procedural description of how to perform that step, and typically ends with one or more output artifacts that serve as input to the next step. This should scaffold the student's learning of concurrency concepts and their application. The steps proposed are essentially as follows, to be applied to a description of a programming problem that in a natural way involves concurrency: a simulation, a program with efficiency requirements or a program with responsiveness demands.

(1) *OO structuring of the problem domain* – Possibly a (sometimes given) sequential program using these classes that shows the essential behavior.
(2) *Adding the concurrency* – Analysis whether concurrency is needed, and what type of concurrency is involved. Decision of which activities should be performed concurrently, which determines the number and roles of threads.
(3) *Analysis on race conditions* – The program is analyzed on possible race conditions due to shared variables using an enhanced UML activity diagram as shown in Figure 1. We enhanced the standard UML activity diagram by explicitly showing shared objects with attributes accessed from multiple threads. For further detail and explanation of this diagram, see [1].

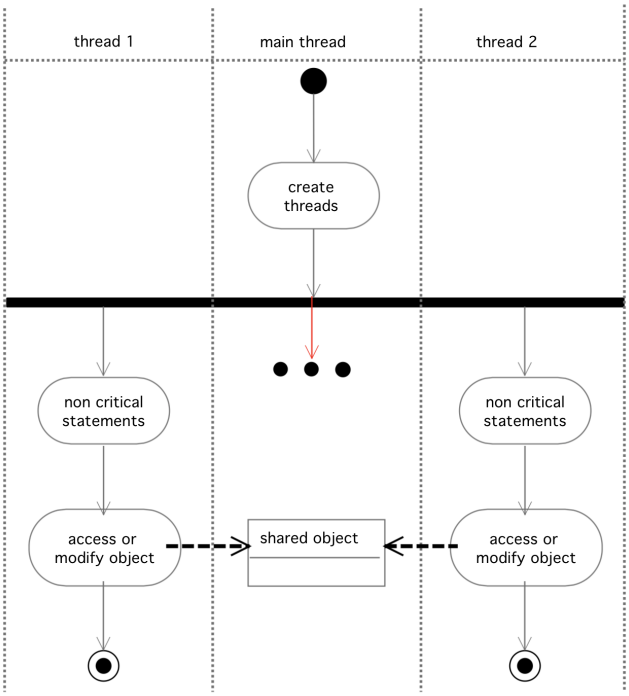

Figure 1: Enhanced activity diagram

If needed, synchronization is applied to regulate all shared variable access.

(4) *Analysis on check-then-act race conditions* – Analysis of the program for check-then-act race conditions, again using the enhanced UML activity diagram, and preventing these using synchronized methods.
(5) *Reflection* – Assessing the quality of the resulting program. Reflection on analysis and design decisions made and going back to the appropriate step if remedy is required.

The artifacts here are the description of the problem, the class diagram, and the enhanced activity diagram. Furthermore, there is a more advanced part of the Steps Plan, concerning deadlocks, this is also not addressed in the present paper.

Our research questions are the following.

(1) What problems did the students encounter in direct relation to the Steps Plan, i.e., problems that can be related to direct deficiencies in steps or combinations of steps?
(2) What problems remain after providing the Steps Plan, i.e., what problems where encountered that are not so much deficiencies of the Steps Plan but more general issues with concurrency education?

Following up on this, we propose improvements, both in the Steps Plan and for the more general issues identified. Among the latter there also emerged matters impacting university education in a broad sense, such as the use of adaptive teaching materials and the balance between the number of subjects treated and the intensity with which they are practiced.

Our paper is organized as follows. Section 2 introduces the method used to investigate the application of the Steps Plan. Section 3 contains the analysis. We formulate the conclusions of the study and suggest improvements to our teaching method and plans for future research in Section 4.

2 METHOD

The Step Plan was tested at three universities during a freshman course on object-oriented programming in Java. The population was heterogeneous, with students from different academic levels and age-groups (varying from first-year bachelor students to students having a professional background in IT enrolled in a bridge program preparing for a master software engineering). The topic concurrency and the Steps Plan were taught and trained during two lectures of two hours and a tutorial of three hours.

In [1], a broad evaluation based on this large number of students indicated that teacher reports and exam results give reason to believe that providing students with this kind of systematic scaffolding is highly beneficial.

For the present paper, in order to extend our understanding of the learning process involved, we have embarked on a more in-depth validation program, based on a subset of these students for which we collected more information. Ten students were observed during a practical work session while making an assignment on concurrency, on which they worked in pairs. The students were expected to complete this assignment in approximately 2 hours. The in-depth evaluation is based on recorded observations of these ten students' work and dialogues, as well as in-depth interviews. The results hereof are reported in the present paper. The research approach we followed is based on [2].

session	duration	text	type
A	196 min	9137 words	Audio
B	51 min	842 words	Video+Screencasts
C	123 min	2188 words	Video+Screencasts
D	185 min	15413 words	Audio
E	32 min	3527 words	Audio

Table 1: Length of sessions

The assignment

The participants' activities were recorded while they were working on the following exercise:

> Imagine that a certain number of travelers (between 60 and 90) arrive at a train station. These people must continue to their final destination, a holiday resort, by taxi. Four taxis are available: two with a capacity of four and two with a capacity of seven people. The taxis ride back and forth as long as there are still people waiting at the station. Each taxi transports as many people as it can, or possibly less, depending on the actual number of people still waiting.

The goal of this assignment was to implement a concurrent simulation of the system that imitated reality as faithfully as possible. For this purpose, in order to reduce the task's size and complexity, the students were given a sequential solution that needed to be converted into a concurrent version. We expected that the students would need about two hours to complete this assignment.

Data collection and analysis

The activities of five different pairs were recorded. This has happened in different ways. Only sound recordings were made of three groups. Video recordings were made from other two groups, combined with screencasts. The audio recordings were transcribed verbatim by the research team. The student activities in the screencasts and video recordings were described in text documents. First, all transcripts and descriptions were read in their entirety. Then, the first set of codes was generated through open pair coding. The main reason for pair coding is to obtain a consensus of two people for all decisions. Similar codes and quotes were clustered and labelled, and categories emerged from this process. Together, the research team organized the codes and clustered them into smaller thematic groups on a plenary team meeting. The discussions during this meeting led to a consensus on the division into themes. This axial coding process was performed using sticky notes on flip charts.

Results

The recorded sessions differed in duration; see Table 1. Table 2 summarizes the result of the coding process. The left column (named *category*) is the result of the axial coding (resulting in *themes* and corresponding *issues*) whereas the right column (named *observations*) shows some characteristic examples of codes that were placed in the same thematic group.

In the next section we will investigate these issues in depth.

3 ANALYSIS

By analyzing the observation results, we identified four shortcomings of our procedure. Two of these are shortcomings concerning an existing step, namely:

- We provided a sequential solution as a step towards a concurrent version. This confused some students, as they were unable to separate the program units to be reused in the concurrent version from those merely present for the purpose of sequential simulation.
- The steps are of too high a level, so that students, while aware what had to be done, were unable to perform the steps. Therefore the big steps have to be split into micro steps, and their execution needs to be practised.

The other two are points of interest missing in the procedure as a whole:

- The exercises do not specify the desired amount of concurrency.
- There is no explicit step requiring the students to check the output for correctness. This would also involve specifying precisely what correctness consists of.

Additionally, we make two observations about the use of anthropomorphism and the role of examples. These issues are discussed in more detail in the remainder of this section.

3.1 Taking a sequential solution as a step towards a concurrent version

We observed that students had difficulties identifying actors. Our procedure gives no concrete instructions on how to determine which activities should be performed concurrently. Moreover, in our exercise a sequential solution to the problem was provided, simulating concurrency through a *step* method. Students often persisted in maintaining parts of this 'step' function, leading to inadequate definitions of active classes, as illustrated by the following observation: one pair of students created a *Taxirunner* thread, without removing the *step* method, that was then started. They studied the output of the program and doubted the solution. However, they did not get the idea to abandon the use of the *step* method.

One explanation for this behavior is that novice programmers tend to be reluctant to modify code that they didn't write themselves, either because they don't understand it properly, or because they are afraid of breaking something that was already (close to) working. Another reason might be a deficiency in the student's conception of threads. Transforming a sequential program into a concurrent one, is by no means straight forward as an exercise for a first level course.

To remedy these issues, the assignments should indicate the desired behavior (e.g. the granularity of each task) more explicitly. Meanwhile, our procedure needs to more explicitly describe how students should decide which actions can be achieved concurrently. Therefore task definition (via the active class pattern) and task creation (using threads) have to be taken into account separately, avoiding any confusion of these activities.

category	*observations*
Concepts	
Synchronization	Synchronizing the wrong code; synchronizing too much or too little code.
Shared resources	Identifying the wrong object as shared.
Race conditions	Misunderstanding the concept of race condition and/or of check-then-act.
Correctness	
Testing	Assuming that concurrent programs are deterministic and that tests are reproducible.
Input/Output analysis	Assuming the program is correct once it produces some output.
Procedural guidance	
Active Class design pattern	Misunderstanding the design pattern: failure to identify actor; referring to the design pattern at the wrong time.
Following the steps	Starting a next step before the preceding one has been completed; taking the steps in the wrong order.
Performing the steps	Confusing the active class with the class that creates the thread; being unable to perform the refactoring required by a correctly identified check-then-act situation; unclear how the domain model classes correspond to the thread model.
Implementation	
Emphasis on code	Inspecting code rather than design; ignoring the procedure and starting on the code right away.
Sequential simulation	Reproducing the limitations of the sequential simulation in the problem statement; trying to adapt the sequential simulation rather than designing afresh.
Other (unexpected) activities	
Anthropomorphisms	Ascribing human motives to threads and objects; detection of final state by observing prolonged inaction.
Unsuccessful approaches	Trial and error, (random) googling; having no plan at all; just responding to IDE error messages

Table 2: Code categories

3.2 Micro steps: the actual procedure is too high level

The suggested procedure is described at a high level of abstraction. In that sense, it is like a recipe that contains instructions like 'prepare chicken stock' without further explanation.

For instance, when a check-than-act situation is identified, the procedure states that 'the actions of checking and acting must occur in the same synchronized method'. In order to bring this about, refactoring steps will be necessary and it is assumed the student knows how to perform these. But it turns out that this is not always the case. If refactoring is done inadequately, this will lead to synchronized blocks that are far too large and take away most of the concurrency, resulting in programs that are correct but of low quality. This is not easily noticed by students [14]. The following fragment shows a example:

> **Student A:** *The problem is you can't put synchronized outside otherwise only one taxi will operate, that is not the intention of course. So you actually want to synchronize a block inside the while loop.*
>
> **Student B:** *If we synchronize this wrong, it still works, right? It should work, right? It is not just that efficient. You don't like one taxi to do all the things.*

A comparable problem occurs when the threads have to be created. One of the misunderstandings that occur here is that the active classes (those that implement *Runnable*) should have the responsibility for creating the threads. Another mistake we have seen is that far too many threads are created, namely a fresh one every time an object has finished some process step.

> **Student:** *The thread has to be created afresh every time. I just happen to know that. [. . .] There are four taxis and there will never be more. But each taxi is inserted into a thread as a task, and when it is finished its work it should go for a new ride. Then you should start a new thread, hence also create one.*

These are situations where students know which step of the procedure needs to be executed but lack sufficient competence to do so. There should be more detailed explanations available of what is to be done within each step of the procedure.

However, filling the teaching materials with detailed explanations of micro-steps will make them extremely boring for students that already have a good understanding of the required actions. In practice, some of the students entering the course will have had little exposure to programming, while others have several years' professional experience. It is impossible to provide a linear text that equally satisfying to both groups.

A possible solution may be found in the concept of adaptive hypertext [3]. This allows for details to be suppressed or provided according to the needs of the student. The decision may be taken explicitly by the student, by clicking some 'expansion' symbol, as has been already implemented by some online newspapers. Alternatively, the decision could be made automatically based on the results of the student's performance on formative tests.

3.3 Desired amount of concurrency

Once the decision has been taken that the program will have to execute some steps concurrently, the question naturally arises what will be allowed to take place concurrently and what not. For instance, a printer should be accessible to multiple threads containing print instructions, but characters produced by different threads should not be arbitrarily intermingled.

Initially, we provided exercises that did not explicitly specify the desired amount of concurrency: we assumed that this would be inferred from the intended use of the program in real life. However, this proved to be a mistake, as many students' submissions (from the population evaluated in [1]) contained suboptimal choices in precisely this area. Hence, we now feel that either the exercise text itself must explicitly specify what is to be performed concurrently, or the students have to be trained in providing such specifications themselves.

The amount of concurrency has three aspects: the number of threads, the granularity of the protected code blocks, and the mutual synchronization of these blocks. The problem with the number of threads has already been addressed in Subsection 3.1. The granularity problem involves determining the size of a critical section: a code fragment that may refer to a shared resource that should not be accessed by different threads at the same time is called a *critical section* [4]. The size of the code fragment is important, because too large a critical section will result in threads being blocked unnecessarily long, thus reducing the advantages of concurrency. The mutual synchronization is important, because synchronizing blocks that do not interfere unnecessarily reduces concurrency, whereas not enough synchronization may lead to unsafe situations.

An extreme instance of a struggle with granularity choice that we observed was students protecting a block containing sleep statements.

Student A: *Why don't you make the whole thing synchronized?*

Student B: *Because the synchronized part should not be made too large.*

Student A: *What's too large?*

Student B: *You should not sleep within the synchronized block. Because there may be no people waiting at the station.*

Student A: *Let's measure how long the sleep lasts.*

A distinct but related problem occurs if too many blocks are synchronized causing too many threads to be blocked, including ones that are not about to access the shared resource. The latter problem occurs when a single lock is used to delay all threads at the critical section, regardless of what resource is actually wanted in a thread's current context. To see an example of this, consider a system for reserving airplane seats: to prevent double bookings, a seat should be blocked during such a transaction. Using a separate lock for each seat will solve this problem efficiently; using a single lock for all seats, although safe, will hamper progress quite unnecessarily.

The opposite effect, using too many different locks, will result in an unsafe solution. Two students observed that method *takePassengers* in class *Taxi* influenced the attribute *numberOfPassengersWaiting* in a different class, and reasoned as follows:

Student A: *Well, it's time to start synchronizing some methods. This method takePassengers will be called by different threads.*

Student B: *We solved that by making it synchronized.*

Student A: *Now that we've done that, I wonder whether we should also synchronize getNumberOfPassengersWaiting.*

Student B: *But that would synchronize all the code within that method.*

Student A: *Right, so we need only synchronize the top level.*

Student B: *It might still be the case that we synchronize too much code.*

Student A: *In principle we have only synchronized takePassengers.*

Here the students seem aware of the danger of blocking too much, but do not realize that by marking the method **synchronized** without explicitly mentioning a lock, they are implicitly using a separate lock for each (taxi) thread; thus different taxis are not prevented from accessing *numberOfPassengersWaiting* simultaneously.

3.4 Correctness

We found two issues concerning correctness, namely students often assume a program is correct once it produces some output, and students are often not aware that because of non-determinism different correct program outputs may result.

First, instead of comparing program output to the requirements in the exercise description, students often assume a program is correct once it produces some output. One example of this way of thinking is shown in the following fragment: students want to ask a question and, in the meantime, execute the program developed so far. They see that many taxis are created, but all these taxis do not transport any passenger. The teaching assistant appears. The students show the output and ask the assistant: 'Is this output correct?'.

In this fragment, the students are considering obviously incorrect program behavior as 'possibly correct', although as long as there are passengers to transport, taxis should not be empty.

Another example fragment is the following:

Student A: *While not station is closed, ... well, ...But, in that case he should close. The train will close the station ... Look at this!*

Student B: *All passengers have been transported.*

Student A: *I think it is ok so. We finished the job. We have to write our report.*

These students are assuming the program is correct because it produces some superficially correct output. But they did not inspect the output precisely, i.e., were all taxis always transporting the maximum number of passengers possible. Neither did they check on anomalous behavior such as halfway vanishing or materializing passengers as might occur due to race conditions.

These findings, i.e. that students are satisfied with a program that compiles and produces some output, correspond with Kolikant [13].

Second, students are not aware that, because of non-determinism in the scheduling of concurrent program parts, different correct outputs may result when running a program several times with the same input. The following fragment shows an example:

Student: *1 2 3 4 1 2. Hey! How is that possible? That is strange. Why didn't it do that a moment ago?*

Here, the student is surprised that the concurrent program produces an output different from the previous execution. This indicates a lack of understanding of concurrent execution and/or an inability to apply the conceptual knowledge concerning non-determinism into real situations. They are looking for *the* correct output, but several correct outputs are often possible.

Apparently, students do not have a notion of what correct behavior is in the multi-threaded application area, and when a program can be assumed to be correct. Observing that there is output is not enough. Instead, the output must be analyzed in relation to the informal specification of the program's behavior as (should be) described in the assignment. It should be noticed that proving a concurrent program is correct is beyond the scope of a freshman course on object-oriented programming. But analyzing and checking on some behavior properties is certainly possible.

In the think-aloud sessions we did not find any fragment where students talked in terms of correctness.

Looking again at our teaching material and exercises, we find that these do not discuss how to draw up informal specifications of program behavior and how to use these to reason about the correctness of a concurrent program. This corresponds to Goetz's observation [9]: '... we often don't write (adequate) specifications ...'. As a result, it is actually impossible to know whether a program is correct. Drawing up informal specifications of expected program behavior and using them in reasoning whether a concurrent program is correct should be clarified at the appropriate level for an introductory course.

This indicates the need for identifying properties that the behavior should have. For example, in case of our train-taxi-passenger exercise, one could require that the number of trains, taxis, and passengers should remain constant during program execution. Or, depending on the goal of the simulation, instead of simply counting passengers at the start and end of a program run, one could check on the passengers' IDs, because due to race conditions, one passenger could disappear while another is created.

In the case of simulation, a natural candidate for a condition on behavior would be an invariant property. Invariants and other conditions could be provided with the exercise, or, for more advanced students, be required to be constructed by the student from the exercise description.

The current Steps Plan has a step, Reflection, that concerns the correctness of the final multi-threaded program, but this step incites only to reflect on the analysis and design decisions made, not how how this analysis should be performed. As a result, the Step Plan should be extended in two ways:
First, an activity analysing/drawing up correctness properties should be added in terms of, e.g., invariants. This can be done at the start of implementing the program, but properties can be further refined when steps towards the final version are being made. Note that formalization of the conditions is beyond the scope of an introductory course: the approach by Felleisen is an inspiring example of how to incorporate explicit condition at this level [8].
Second, at several points in the step plan, the student should be guided to reflect on how the program satisfies such properties. In particular the step Reflection should be extended with means how to use the properties to determine whether the final program behaves correctly.

Additionally, despite the explanation of the concept of non-determinism as part of the Steps Plan, it turns out that students are not aware of this concept in practical situations, i.e., that a correct concurrent program can produce several different outputs due to non-determinism in the scheduling of the concurrent program parts. This impacts the notion of correct behavior: students should be aware which differences in output over different runs are an indication of incorrect behavior and which are due to acceptable non-determinism induced by concurrency. This difference should receive more attention in the teaching and teaching materials. Also, in exercises for the initial stages, to scaffold students, concurrency-induced non-determinism should be made explicit in the specification. For example, the description of correct behavior in the train-taxi-passenger exercise could explicitly allow different orderings of the taxis in different runs. Next to the explanation and the use of this concept in complete exercises, the application of this concept should be practiced in smaller exercises, so called part-tasks [19], to train this routine aspect up to a higher level of understanding and automation.

3.5 Anthropomorphism

Anthropomorphism is the attribution of human characteristics to things that are not human. Analyzing the think-aloud sessions, we observed, as a fortunate by-catch, that students often use an anthropomorphic style in talking about objects and their behavior. For example:

Student: *Voilà, out of the box. O yes,* he *is going to create taxis first, then there arrives suddenly a train with 85 passengers. The taxis take all these passengers ...*

Here, *he* points to the part of the program responsible for creating taxi objects.

In literature, anthropomorphism is considered as an innate tendency of human psychology [15]. It is known that anthropomorphism can help in thinking and talking about difficult and unfamiliar topics [11]. Nevertheless, many scientists disapprove the use of anthropomorphism: It is considered as a symptom of professional immaturity, i.e. it disguises that one's knowledge is insufficient [5, 6, 11, 17].

In our recorded sessions, we found passages where the use of anthropomorphism is just a style of talking that does not seem to hinder understanding. The passage above is an example of this. Anthropomorphism may hinder understanding, however, when programming objects are attributed human characteristics that can not and will not be implemented. Then the student is constructing a programming model that does not comply with reality and will hamper his understanding and programming abilities.

The following fragment is an interesting example where this case may be applicable.

> **Student:** *Yes, that is wrong. There should be something … indeed. […] Can you say that after a number of taxi rides he simply stops? Or, that after a long time of waiting, in case he has waited ten times and there still aren't passengers there, he goes home?*

In this example, a final state of a taxi is observed by a prolonged inaction of that taxi. Where a human would be tired to wait, it is obviously not the reason for a thread to stop. How mistaken this anthropomorphism is, remains the question, though. Remember that we have a simulation here where the student's program is intended to simulate human behavior (human passengers, taxis driven by humans). From the exercise description, it is not so clear what exactly is simulated. Is it just a logistic procedure, implemented by the taxi drivers and not intended to be influenced by human considerations such as tiredness or are we simulating human behavior here?

In our opinion, anthropomorphic thinking can be helpful to explore a new subject or to get to grips with a new problem. It can, for instance, be used in an OO simulation game, resembling the well-known CRC modeling technique [10], performed by students to explore the way objects behave and interact with each other. However, if this is carried out too far, there is a risk that students will set themselves on the wrong track. In the case of simulations where humans are involved, extra care has to be taken to make clear to the students what behavior is to be simulated, e.g., behavior of an inanimate algorithm, possibly implemented by humans, or decisions made by humans. This may help the student to not confuse the innocent anthropomorphisms and the helpful anthropomorphisms of the phase of getting to grips with the problem and the possibly detrimental anthropomorphisms regarding, e.g., programming objects.

3.6 Unrealistic examples

Many instructors have found it difficult to construct high quality programming assignments. Several authors have looked at this issue and proposed different criteria which should be taken into account when developing programming assignments. For instance, Falkner and Palmer [7] state that a good programming assignment should be based on a real-world problem and allow the students to generate a realistic solution.

In our opinion, the three main reasons to introduce concurrency into a program, are improvement of efficiency and processor utilization, avoidance of nonresponsiveness, and simulation of inherently parallel processes in the real world.

It is relatively simple to give examples and assignments for the first two cases that are both practically relevant and relate to a realistic problem. If we look at our step-by-step plan, we find that provided procedural guidance mainly focuses on the third category of applications: the simulations. In practice, simulations are used to determine properties of processes by modeling these processes as a computer program and then analyzing the results of this program. However, often it is not necessary to use concurrency in such a model. On the contrary, in many cases a sequential implementation appears to work much better. This also seems to be the case in the taxi assignment used in this study. The simulation in this assignment has no realistic goal. In fact, the taxi problem is only used as a metaphor to visualize the abstract concurrency concepts. The need to make explicit use of concurrency for this purpose is lacking, as a result of which students are often unable to discover the constructs intended by the lecturers themselves. This could explain why students stubbornly stick to the sequential solution that was provided as a starting point.

4 CONCLUSIONS

Using the Steps Plan in actual education has confirmed our initial intuition that something along these lines would be useful as a guide providing scaffolding for those students that have no experience how to attack concurrency exercises. The observation results do not show a reason to change the choice or the order of the macro-steps in the Steps Plan.

Students have to deal with amount of concurrency, correctness, race conditions and synchronization. Problems observed pose the question how to help the student make these things concrete and explicit, and how to provide procedural help. A general observation is, that this cannot be provided by the Step Plan alone: exercises need to be crafted that provide detailed explicit scaffolding at the beginning and gently evolve to posing more demands on the independent thinking of the student.

Analysis of the difficulties students ran into points the way to improvements.

- Providing a sequential solution to a problem for which a concurrent solution is sought is not helpful. If we want to assist students to get started, a framework of domain classes is preferable.
- Exercises should specify clearly which activities are supposed to proceed concurrently. If this decision is left to the (more advanced) students, they ought to provide explicit reasoning leading to their choice.
- Task definition (via the active class pattern) and task creation (using threads) have to be taken into account separately in the Steps Plan, avoiding any confusion of these activities.
- Detailed information in terms of micro-steps on how to perform the macro-steps in the Steps Plan should be available

on demand: the role of adaptive hypertext needs to be investigated.
- The step Reflection of the Steps Plan should be extended with means of how to check the output for correctness. This requires an analysis to determine which correctness properties would be visible from the output, taking into account the non-deterministic character of concurrent programs.
- Examples and assignments should be both practically relevant and relate to a realistic problem. This is more often the case with problems addressing efficiency and responsiveness than with simulation of real-world parallel processes. Exercises should be adapted to this insight.
- Anthropomorphic thinking turned out to be helpful when thinking about a problem domain, but turned out to be dangerous when applied to, i.e., programming objects. We should make our students and teachers aware of this.

Two aspects of our research that have not yet been discussed are the following.

First, the detailed observation of students (through screencasts and audio recordings) and the comprehensive analysis of the generated data leads to valuable knowledge about the way students learn to program. This information remains invisible if we solely base conclusions about the learning behavior of students on the assessment of delivered products and disregard the way in which students proceeded. In our specific case of the Steps Plan, this has yielded a number of concrete points for improvement that would otherwise have gone unnoticed.

Second, while working on the assignment, students sometimes seemed not to make any progress at all. We observed that students often had difficulty with concepts that had already been introduced and that the teachers assumed students would master. Due to the cognitive load [18] that this creates, the entire development process seems to be stagnating. We think that for learning new concepts, students must have sufficient time and opportunity to practice. To create room for this, teachers should consider reducing the number of subjects in a course.

ACKNOWLEDGEMENT

This project is part of the research group Didactics of Informatics and is carried out in collaboration with E. Barendsen. Participating universities in this project are the Radboud University, Eindhoven University of Technology and Open University of the Netherlands.

REFERENCES

[1] A. Bijlsma, C. Huizing, R. Kuiper, H. J. M. Passier, H. J. Pootjes, and J. E. W. Smetsers. A structured design methodology for concurrent programming. In *Proceedings of the 6th Computer Science Education Research Conference*, CSERC '17, pages 1–9, New York, NY, USA, 2017. ACM.

[2] Louis Cohen, Lawrence Manion, and Keith Morrison. *Research methods in education.* London, New York: Routledge, 2013.

[3] Paul De Bra. Adaptive educational hypermedia on the web. *Commun. ACM*, 45(5):60–61, May 2002.

[4] E. W. Dijkstra. Solution of a problem in concurrent programming control. *Commun. ACM*, 26(1):21–22, January 1983.

[5] E.W. Dijkstra. How do we tell truths that might hurt, 1975. EWD 498.

[6] E.W. Dijkstra. On anthropomorphism in science, 1985. EWD 936.

[7] Katrina Falkner and Edward Palmer. Developing authentic problem solving skills in introductory computing classes. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, SIGCSE '09, pages 4–8, New York, NY, USA, 2009. ACM.

[8] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. *How to Design Programs: An Introduction to Programming and Computing.* MIT Press, Cambridge, MA, USA, 2001.

[9] Brian Goetz, Tim Peierls, Joshua Bloch, Joseph Bowbeer, Doug Lea, and David Holmes. *Java Concurrency in Practice.* Addison-Wesley, Upper Saddle River, New Jersey, 2006.

[10] Lars Hvam, Jesper Riis, and Benjamin Loer Hansen. Crc cards for product modelling. *Computers in Industry*, 50(1):57 – 70, 2003.

[11] M. Kallery and D. Psillos. Anthropomorphism and animism in early years science: Why teachers use them, how they conceptualise them and what are their views on their use. *Research in Science Education*, 34(3):291–311, Sep 2004.

[12] P.A. Kirschner, J. Sweller, and R.E. Clark. Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational psychologist*, 41(2):75–86, 2006.

[13] Yifat Ben-David Kolikant. Students' alternative standards for correctness. In *Proceedings of the First International Workshop on Computing Education Research*, ICER '05, pages 37–43, New York, NY, USA, 2005. ACM.

[14] Gary Lewandowski, Dennis J. Bouvier, Robert McCartney, Kate Sanders, and Beth Simon. Commonsense computing (episode 3): Concurrency and concert tickets. In *Proceedings of the Third International Workshop on Computing Education Research*, ICER '07, pages 133–144, New York, NY, USA, 2007. ACM.

[15] K.J. McCaffree. Is magical thinking good?. *Skeptic*, 19(1):59 – 61, 2014.

[16] T. Plomp. Educational design research: An introduction. In T. Plomp and N. Nieveen, editors, *Educational design research*, pages 10–51. Enschede: SLO, 2013.

[17] L. Vaillant. Anthropomorphism gone wrong: Poor motivating example for oop, 2015.

[18] Jeroen J. G. van Merriënboer and John Sweller. Cognitive load theory and complex learning: Recent developments and future directions. *Educational Psychology Review*, 17(2):147–177, Jun 2005.

[19] Jeroen J.G. van Merriënboer, Richard E. Clark, and Marcel B.M. De Croock. Blueprints for complex learning: The 4c/id-model. *Educational Technology Research and Development*, 50(2):39–61, 2002.

[20] Jeroen J.G. van Merriënboer and Paul A. Kirschner. *Ten Steps to Complex Learning, a systematic appraoch to four-component instructional design.* Taylor & Francis, New York, NY, USA, second edition, 2013.