Degree Project in Computer Science and Engineering,

First cycle, 15 credits

Student understanding of Go, Concurrency and Parallelism

Analysis using Concept Maps

ALEXANDRE VILANOVA

Stockholm, Sweden 2022

Student understanding of Go, Concurrency and Parallelism. Analysis using Concept Maps

ALEXANDRE VILANOVA

Bachelor in Computer Science and Engineering
Date: June 8, 2022
Supervisor: Richard Glassey
Examiner: Pawel Herman
School of Electrical Engineering and Computer Science
Swedish title: Elevernas förståelse för Go, Samtidighet och Parallellitet - Analys med hjälp av Begreppskartor

Abstract

In this report we have used concept maps to analyze student understanding of the Go programming language, concurrency and parallelism. We designed two concept mapping exercises, one about Go and the other one about concurrency and parallelism. We sent them to college students after taking a course about parallel and concurrent programming in introduction to computer science and collected their takes. We interpreted the concept maps as directed graphs and analyzed their connections. We were able to perceive some common connection patterns within the concept maps. Most students got the really basic connections right but struggled to differentiate fundamental concepts such as Go data types. We saw different ways of thinking when it takes to concurrency and parallelism concepts.

Sammanfattning

I den här rapporten har vi använt begreppskartor för att analysera elevernas förståelse av programmeringsspråket Go, samtidighet och parallellitet. Vi utformade två konceptkartläggningsövningar, en om Go och en om samtidighet och parallellism. Vi skickade dem till högskolestudenter efter att ha läst en kurs om parallell och samtidig programmering i introduktion till datavetenskap och samlade in deras svar. Vi tolkade begreppskartorna som riktade grafer och analyserade deras kopplingar. Vi kunde uppfatta vissa gemensamma kopplingsmönster inom konceptkartorna. De flesta studenterna fick de riktigt grundläggande kopplingarna rätt men hade svårt att särskilja grundläggande begrepp som Go-datatyper. Vi såg olika sätt att tänka när det gäller samtidighet och parallellitet.

Contents

Chapter 1

Introduction

Parallel and distributed computing has become a *core* area in computer science majors. However, introducing it in early curriculum stages may bring several cognitive and technical challenges that can be particularly difficult to tackle.

This project is a concept map analysis on the learning outcomes of first year computer science students when it takes to the Go programming language, concurrency and parallelism.

Concept maps can be seen as directed graphs where each node is labeled with the name of a concept and each edge is labeled with what describes the relationship between the endpoints.

Encoding concept maps as graphs leads to multiple interesting ways to evaluate knowledge. Graph similarity algorithms and graph properties can be used to find common patterns within the maps.

The results from this degree project can be useful to enhance teaching methodologies when it takes to new programming languages, concurrency and parallelism. It also provides new insights about the use of concept maps as a tool to evaluate understanding.

1.1 Context

This work is a a final thesis for the Bachelor Degree in Informatics Engineering (*computing* specialization) from Universitat Politècnica de Catalunya (Facultat d'Informàtica de Barcelona).

It is written, supervised and evaluated abroad at KTH Royal Institute of Technology (School of Electrical Engineering and Computer Science) in Stockholm.

The supervisor for this degree project is Richard James Glassey, member of the KTH Technology Enhanced Learning research group.

The project is related to topics like *computer science education*, *programming languages*, *concurrency and parallelism*, *concept maps* and *knowledge measurement*.

1.2 Research questions

This project aims to provide answers to:

- **RQ1**: What are some common student misconceptions when learning the concepts of Go?
- **RQ2**: How do students perceive concurrency and parallelism concepts in introductory computer science?
- **RQ3**: How does previous knowledge in different programming languages like Python and Java influence student understanding of Go?

1.3 Scope

The research on this project is focused on describing learning outcomes from a group of students regarding the Go programming language, concurrency and parallelism. It is not within the scope of the project to provide a solution or alternative teaching methodologies in any of the analyzed topics.

Chapter 2

Background

In this chapter, information relevant in different parts of the project will be contextualized. These are the topics that will be covered:

- Computer science education.
- Concurrency and parallelism in computer science education.
- Concept maps.
- The Go programming language.

Here it is defined some key terminology used in the report:

CM	Concept Map
CS	Computer Science
DG	Directed Graph
TA	Teacher Assistant
TEL	Technology Enhanced Learning
UI	User Interface

2.1 Computer science education

Computer science education is the sub-discipline of pedagogy that addresses topics like *what should be taught* and *how should we teach this* in the field of CS [1]. It studies the impact of CS in society and the intersections between CS and other disciplines such as philosophy, psychology, linguistics and mathematics.

CS education was born together with modern computing and thus, it is a much younger field if we compare it to mathematics and science education. [2]

2.1.1 Concurrency and parallelism

Now that we have reached the point multi-core processors have become the norm for computers and smartphones, it is important to take advantage of the possible additional computer power by developing optimal software. For this reason, curricula guidelines have been proposed in order to target the complexities that parallel and distributed computing involve [3].

CS2013[1] started recognizing this with a dedicated parallel and distributed computing knowledge area with *core* hours, as well as dispersing parallelism concepts across other fundamental knowledge areas [4].

2.2 Concept maps

Concept maps are a schematic representation of a network of concepts or ideas related by linking words (propositions). They provide learners and instructors alike with opportunities to share, discuss and revise understanding and meaningful integration of concepts. [5]

They can be seen as directed graphs where each node is labeled with the name of a concept and each edge is labeled with what describes the relationship between the endpoints.

Concept maps have been useful to measure knowledge in a wide variety of areas in the past, including computer science [6].

[1]The *Computer Science Curricula 2013* is the yearly update to the undergraduate computing programs guidelines published by the Association for Computing Machinery and the IEEE Computer Society in 2013.

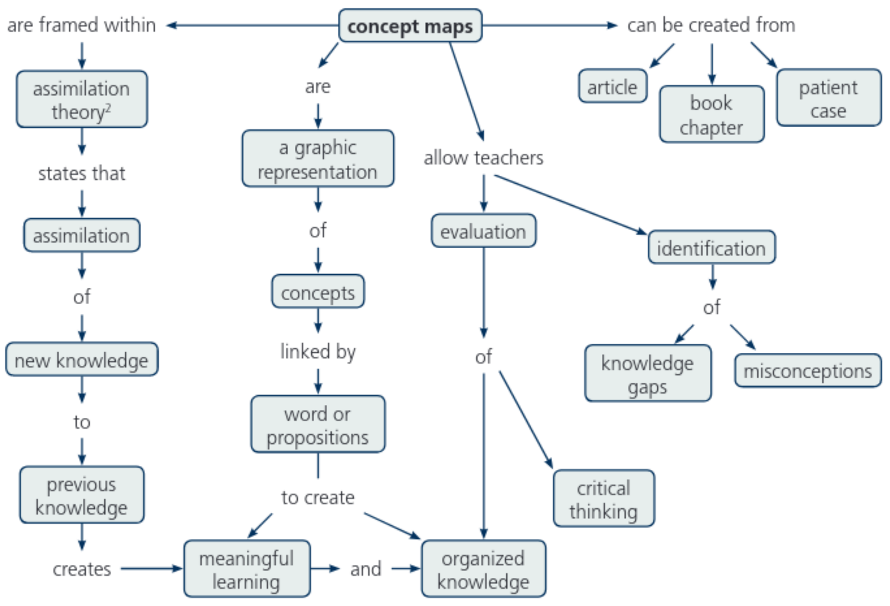


Figure 2.1: Concept Map example[2]

2.3 The Go programming language

The Go programming language was created at Google by Robert Griesemer, Rob Pike, and Ken Thompson in 2009. Ever since it was released, users of traditional compiled languages have found it to be a refreshing change due to it's simplicity and high-quality libraries [7].

It aims to be a great language for systems programming with support for multi-processing and a fresh and lightweight take on object-oriented design. Google described it as an attempt to combine the development speed of working in a dynamic language like Python with the performance and safety of a compiled language like C or C++.

Go provides rich support for concurrency using goroutines and channels:

- A `goroutine` is a function that is capable of running concurrently with other functions.
- The `chanel` type provides a way for two goroutines to communicate with one another and synchronize their execution.

[2]Image source: *Concept Maps: Definition, Structure, and Scoring* [5].

2.4 Previous work

There have been previous studies in the past where the use of concept maps as a method to assess understanding has been useful. The research domains include human resource development [8], biology [9] and training in dental medicine [10].

Computer science and software development have also been fields of study where concept mapping techniques have been applied in the past. There have been studies where concept maps assessments have been used for teaching computer programming [11]. Furthermore, concept mapping techniques have also been used in the field of object-oriented programming [12].

Chapter 3

Methods

In this chapter we show the methodologies used throughout the project. We describe the context and background from the research participants that contributed to the analysis. We list the requirements we considered important when it takes to the data collection and we go over the taken approach. Finally we explain the tools and methods used in the data analysis.

3.1 Research participants

The participants that provided the data used in this research are students from a KTH course named *Parallel and Concurrent Programming in Introduction to Computer Science* that goes by the code DD1396.

In DD1396, students learn about the theory under concurrency and parallelism in a CS background. They also put the theory into practice and learn how to design and implement simple concurrent programs using Go.

The students do not necessarily come from the same academic path. Depending on the previously taken courses they might have already programmed in Java or Python. Given that DD1396 is part of the KTH courses in introductory computer science, we assume most student will be relatively new to programming.

It is important to note that research participants had just finished taking DD1396 the moment the data was collected. In fact, the data was collected just after their final exam.

3.2 Data collection

In order to carry out the data collection for the project, we requested research participants to:

- Solve CM exercises about Go, concurrency and parallelism.
- Answer a short survey about their programming background.

CMs can be represented both in physical formats (e.g. using pen and paper) and digital environments (e.g. using CM editors). When collecting the data from research participants, it is important to keep a consistency in the way CMs are gathered. This makes the analysis process easier while also adds value to the collected data since all participants are being provided with the same tools.

In the case of this project, keeping the data collection in a digital environment seemed more suitable than opting for a physical format. Automating the parsing from a drawn CM to digital formats such as JSON is easy to do when the CMs are collected through a digital platform. It helps saving time when processing the data and it eases the analysis part overall. Moreover, hosting on-site tests is substantially more complicated.

It is important that the platform in which the data is collected is as user friendly as possible. Choosing a CM editor with lots of features would be confusing so it is positive that the chosen editor is as simple as possible.

Requiring to install a program is an unnecessary extra step. Keeping the process web-based and easy to follow is a priority.

To conclude, key requirements for the data collection were the following:

- The platform in which the data is collected is web-based.
- The chosen CM editor has an intuitive and minimalist UI.
- It is possible to parse the collected CM to a digital format such as JSON with no manual work required.

3.2.1 CMTask

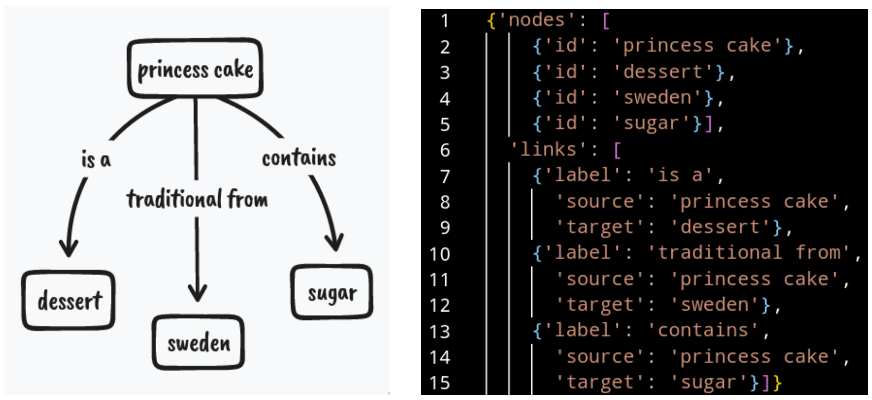
After searching for a while, we could not find any platform that allowed us to host online concept mapping exercises. For this reason we decided to develop CMTask, a simple web-app that helped us do the job.

The chosen stack for CMTask was React for the front-end together with tl-draw[1] as an embedded CM editor, Go for the backend and Digital Ocean on the hosting side.

A wide variety of CM editors were considered but tldraw turned out to be the most appealing. It has embedded support that supports disabling advanced features (not needed for the exercises), it is free, open-source, functional and minimalist. It also has an active community supporting the project.

React was chosen for the frontend because it can easily integrate tldraw. Go was used for the backend because it allows creating a simple web-server in very few lines of code and we were already familiar with the language. Digital Ocean was selected for the hosting because we already had an up and running server which we ended up re-using.

In figure 3.1 there is a toy example drawn in CMTask together with a screenshot showing the corresponding JSON encoding.



```
{'nodes': [
    {'id': 'princess cake'},
    {'id': 'dessert'},
    {'id': 'sweden'},
    {'id': 'sugar'}],
  'links': [
    {'label': 'is a',
      'source': 'princess cake',
      'target': 'dessert'},
    {'label': 'traditional from',
      'source': 'princess cake',
      'target': 'sweden'},
    {'label': 'contains',
      'source': 'princess cake',
      'target': 'sugar'}]}
```

Figure 3.1: CMTask demo map together with the parsed output

3.2.2 Concept map exercises

When it takes to the CM exercises there were two options to consider. The first one was asking students to draw maps from scratch given specific topics. The second one was giving students previously defined incomplete maps about specific topics together with bags of related concepts they were expected to incorporate.

We went for the second option as we thought it would narrow the variance in the results making the whole analysis more interesting. If we had made the

[1]The tldraw project repository can be found at github.com/tldraw/tldraw

analysis on a bigger crowd of participants, the first alternative would have been interesting to consider.

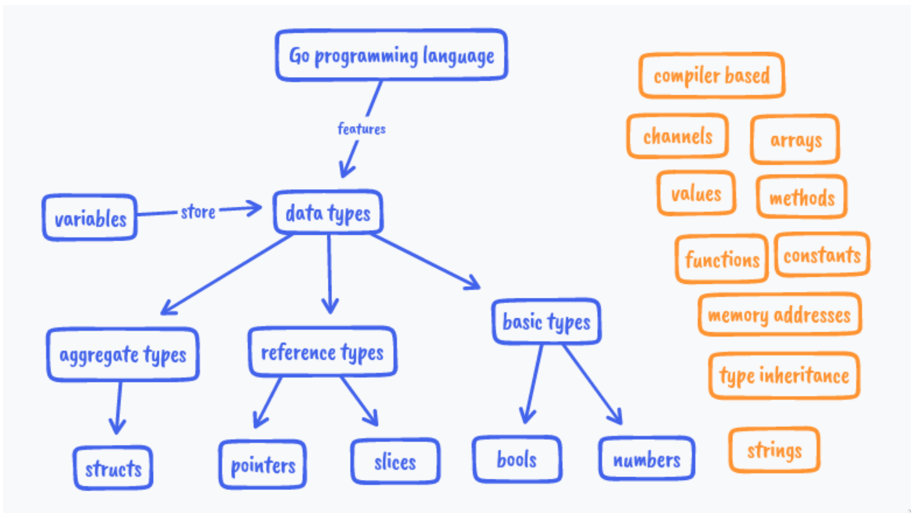


Figure 3.2: Programming in Go exercise statement

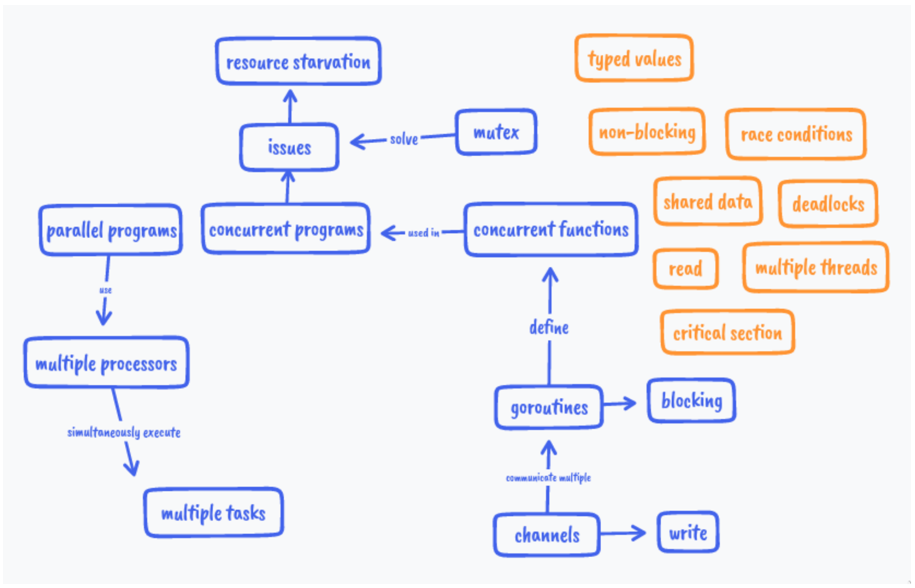


Figure 3.3: Concurrency and Parallelism exercise statement

Find the statements for the Go exercise and the concurrency and parallelism one in figures 3.2 and 3.3 respectively. The concepts from the concurrency and parallelism exercise were taken from DD1396 slides. The concepts from the Go task were taken from both DD1396 slides and the Go programming language documentation[2].

The meaning of the colors in CMTask exercises is the following:

- **Blue**: concepts and arrows that form the pre-defined map. Participants should build on top of it and they should neither delete nor change it's concepts and connections.
- **Orange**: bag of concepts disconnected from the map. Participants are responsible of adding the necessary (black) arrows that link them to the blue concepts.
- **Black**: this is the color participants are allowed to use when adding new concepts and arrows to the map.

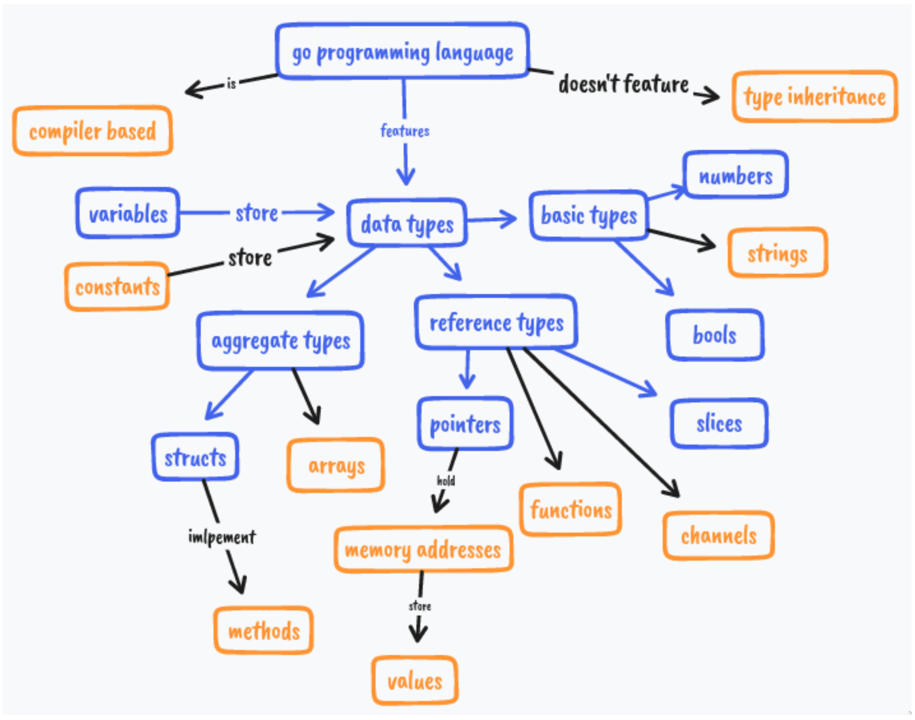


Figure 3.4: Programming in Go exercise solution

[2]The Go programming language documentation can be found at go.dev/doc

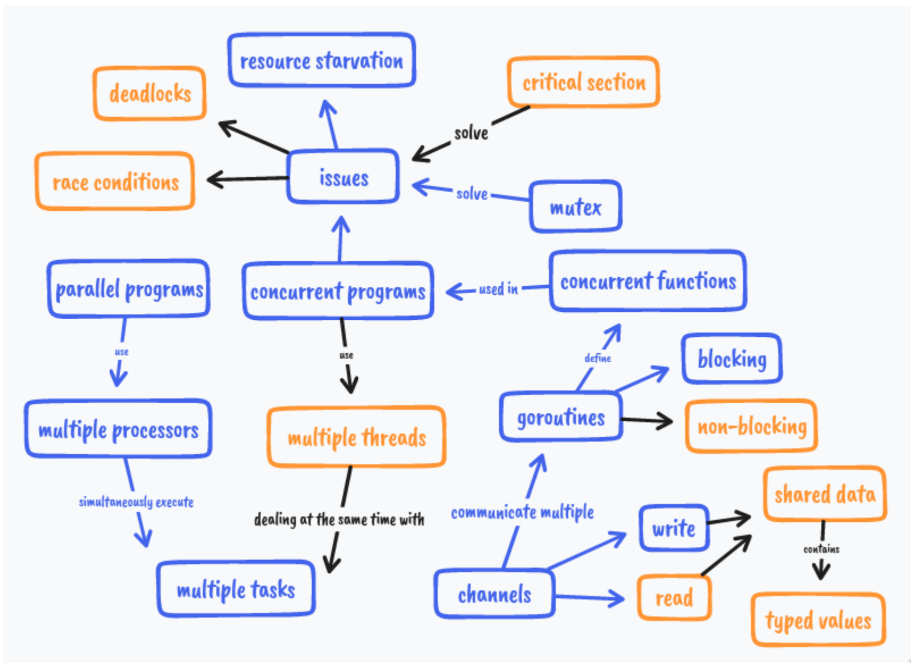


Figure 3.5: Concurrency and Parallelism exercise solution

Find our proposed solutions for the Go exercise and the concurrency and parallelism one in figures 3.4 and 3.5 respectively. Given the freedom provided by the CM format there is not one single *good* solution but lots of them. These are our takes.

3.2.3 Programming experience survey

Running small survey after the two CM exercises was useful to gather information about the programming experience from the research participants. The survey consisted of 5 questions.

The first question was about previously taken KTH courses. As mentioned in section 3.1, not all research participants came from the same academic path. DD1396 organizers provided a list with the relevant previous programming courses their students might had taken. The question in the survey was a simple checkbox where students could check which courses they had been registered to. Find a list of all the possible options in table 3.1.

Code	Title	Language
DD1337	Programming	Java
DD1338	Algorithms and Data Structures	Java
DD1310	Programming Techniques	Python
DD1380	Java Programming for Python Programmers	Java
DD1331	Fundamentals of Programming	Python

Table 3.1: Previous programming courses participants might have taken

The second question was about programming experience in general. The question itself was *For how long have you been programming?*. This question was useful to see the general programming background from research participants. They could have prior extracurricular experience or be self-taught at programming. The possible answers were:

- Less than one year.
- One to two years.
- Three to four years.
- More than four years.

The third and fourth questions were about confidence when using Python and Java respectively. We planned to use this information to find possible relations between the languages students had previously learnt and the CM exercises. The possible answers for both the Python and Java questions were:

- I have never used it.
- I have used it a couple of times.
- I have used it multiple times, I usually use it and I'm confident using it.

Finally, the last question was *What is your level in Go programming?*. We used this question to filter students that already knew Go before taking DD1396. The possible answers were:

- I used it for the first time in DD1396.
- I had already used it before taking DD1396 but not that much.
- I have used it multiple times, I usually use it and I'm confident using it.

3.3 Data analysis

In the data analysis part we used Python together with Jupyter. Some useful Python modules were Pandas, Matplotlib, NetowrkX, GMatch4Py, SciPy and NumPy.

We created a CSV file that contained the answers from the programming experience survey. We used Pandas to read the data set and then Matplotlib to plot the results.

We already had the CMs encoded in JSON files. In order to properly analyze them we needed to parse them to directed graph data structures so we created a custom parser that transforms the JSON graphs obtained from CMTask to NetowrkX directed graphs.

Once we had the CMs interpreted as directed graphs we were able to analyze their properties and get an overall view on the student understanding.

Given that CM exercises have an open-ended set of correct solutions, we did not compare the student maps to our solutions showed in section 3.2.2 or any other kind of predefined *master* map. We approached our research in a analytical way, analyzing what the students actually know instead of ranking their knowledge.

3.4 Limitations

There are some potential inconveniences that can negatively affect the quality of the results.

The first one is the number of participants contributing to the study. Given that the CM exercises and the programming experience survey will be optional extra-curricular tasks, it is likely that some students skip them.

The total number of participants being low can be a problem as very few results may not lead to interesting and meaningful discussion topics.

Finally, the data quality is another important factor to take into account. Participants not being familiar with concept mapping can be a limitation and so can be participants delivering rushed and effortless takes.

Chapter 4

Results

In this chapter we show plots and tables containing the results fruit of the data analysis. We begin giving an insight on the programming background from our research participants. Then we proceed providing the results from the CM exercises. We got data from a total of **53 students** out of the 290 students that were taking DD1396.

4.1 Programming experience survey

In table 4.1 there are all the participants grouped by the set of previous KTH programming courses they have taken. Check table 3.1 to see the information for each course code.

Group	Participants	DD1337	DD1338	DD1310	DD1380	DD1331
1	28	✓	✓			
2	15					✓
3	3		✓	✓	✓	
4	2			✓	✓	
5	1	✓				
6	1				✓	
7	1		✓		✓	
8	1			✓	✓	✓
9	1	✓	✓	✓	✓	✓

Table 4.1: Participant grouped by previous KTH programming courses

The groups are listed in descending order. The first one is made of the 52.83% of the total participation while the second one includes the 28.3%. Groups from 3 to 9 hold the resting 18.87%. We can see how most research participants (81.13%) are within the first two groups.

In figure 4.1 there are 4 plots showing the results from the programming experience part of the survey. All the questions, together with the corresponding answers are explained in section 3.2.3. These plots take into account all the 53 research participants that contributed to the analysis.

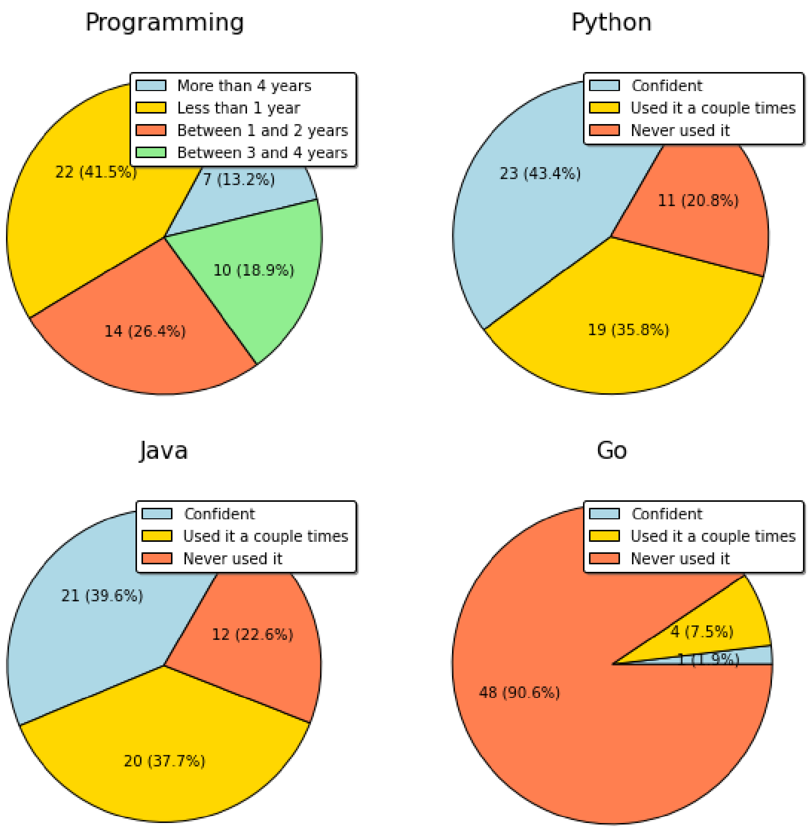


Figure 4.1: Participant programming experience charts

Given that the majority of research participants are grouped within groups 1 and 2, we decided to generate plots and find out which was the programming experience in these groups in particular.

Figures 4.2 and 4.3 contain plots showing the programming experience in both Python and Java from groups 1 and 2 respectively.

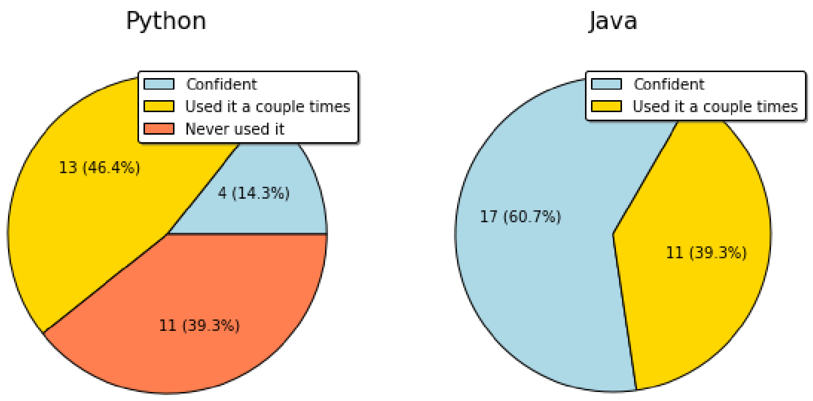


Figure 4.2: Python and Java experience charts from Group 1

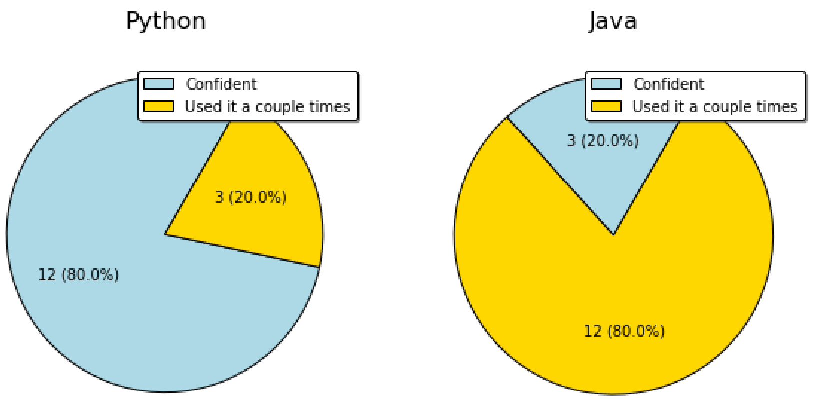


Figure 4.3: Python and Java experience charts from Group 2

4.2 Concept map exercises

In this section we show tables containing connections from both the Go exercise and the concurrency and parallelism one filtered under different parameters. We also provide some of the solutions within the data from the participants.

For the tables in this section it is important to consider the following:

- We took into account the CMs from all the research participants (53 in total).
- The tables include the connections together with the set of edge labels

participants used and the total number of occurrences. In the labels column *""* means *unlabeled edge*.

- Concept names and connection labels in these tables follow the color code defined in section 3.2.2: **orange** concepts are the ones that were disconnected (and for students to connect), **blue** concepts are the ones that were part of the predefined concept map in the statement.

4.2.1 Go programming language exercise

In table 4.2 it is possible to find the most frequent connections in the Go exercise.

Connection			Labels	Occ.
strings	←	basic types	{""}	35
compiler based	←	Go programming language	{"", "is"}	31
channels	←	reference types	{""}	21
channels	←	aggregate types	{""}	8
arrays	←	reference types	{""}	15
arrays	←	aggregate types	{""}	12
arrays	←	slices	{"", "constant size"}	11
arrays	←	basic types	{""}	7
functions	←	Go programming language	{"", "uses", "also features"}	14
functions	→	methods	{""}	12
functions	←	aggregate types	{""}	9
functions	←	reference types	{""}	7
values	→	variables	{"", "given to"}	10
values	←	variables	{"", "has", "store", "stores"}	10
values	→	constants	{"", "given to", "and can be"}	7
memory addresses	←	pointers	{""points to", ", "needs"}	22
memory addresses	←	reference types	{""}	11
constants	←	basic types	{"", "are often"}	13
constants	→	data types	{"", "store"}	12
methods	←	Go programming language	{""}	9
type inheritance	←	Go programming language	{"", "has"}	8
type inheritance	←	structs	{""}	8

Table 4.2: Most frequent connections in the Go exercise

Tables 4.3, 4.4, 4.5 and 4.6 show the most frequent connections when it takes to concepts "array", "channels", "functions" and "type inheritance" respectively in the Go exercise.

Connection			Labels	Occ.
arrays	←	reference types	{""}	15
arrays	←	aggregate types	{""}	12
arrays	←	basic types	{""}	7

Table 4.3: Most frequent connections for "array" in the Go exercise

Connection			Labels	Occ.
channels	←	reference types	{""}	21
channels	←	aggregate types	{""}	8
channels	←	basic types	{""}	6

Table 4.4: Most frequent connections for "channels" in the Go exercise

Connection			Labels	Occ.
functions	←	Go programming language	{"", "uses", "also features"}	14
functions	→	methods	{""}	12
functions	←	aggregate types	{""}	9
functions	←	reference types	{""}	7

Table 4.5: Most frequent connections for "functions" in the Go exercise

Connection			Labels	Occ.
type inheritance	←	Go programming language	{"", "has"}	8
type inheritance	←	structs	{""}	8
type inheritance	←	data types	{""}	5

Table 4.6: Most frequent connections for "type inheritance" in the Go exercise

Figures 4.4 and 4.5 show two examples of solutions for the Go exercise provided by two of the research participants.

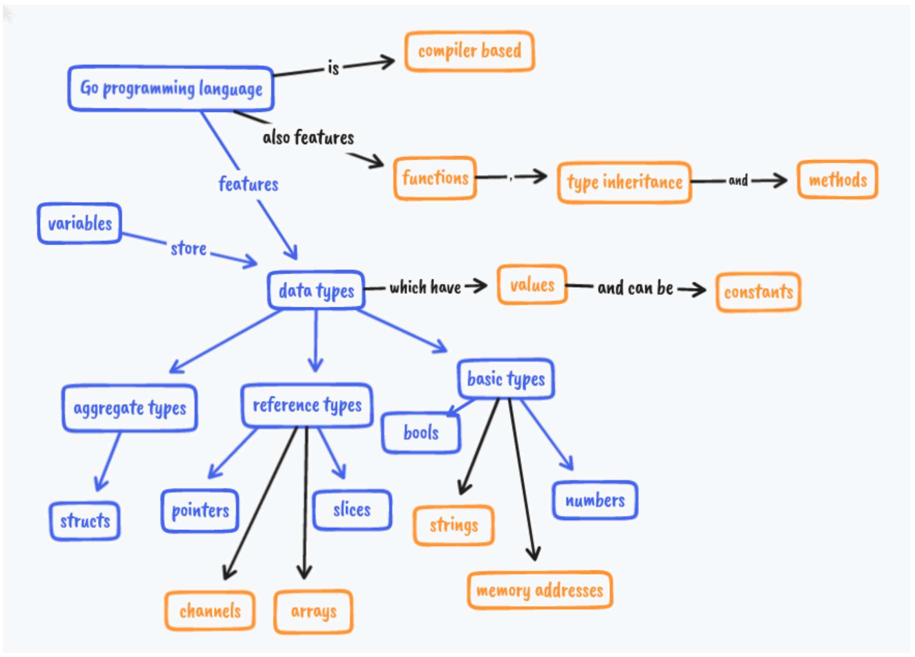


Figure 4.4: Go exercise solution (example 1)

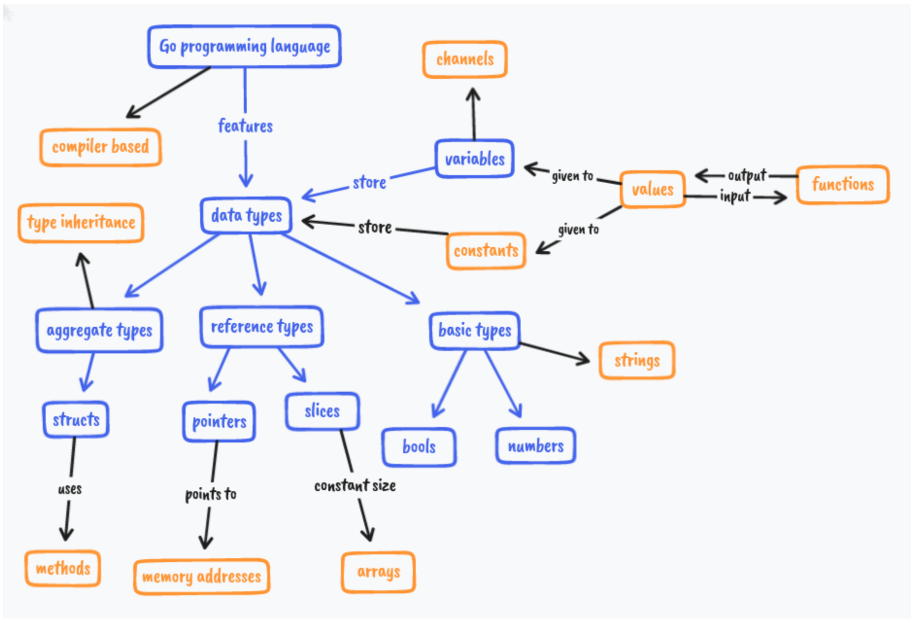


Figure 4.5: Go exercise solution (example 2)

4.2.2 Concurrency and parallelism exercise

In table 4.7 it is possible to find the most frequent connections in the concurrency and parallelism exercise.

Connection			Labels	Occ.
read	←	channels	{""}	39
read	→	channels	{""}	7
deadlocks	←	issues	{""}	28
deadlocks	←	blocking	{"", "may lead to", "causes"}	9
deadlocks	←	goroutines	{"", "can cause"}	8
race conditions	←	issues	{""}	24
race conditions	←	shared data	{"", "causes"}	5
non-blocking	←	goroutines	{""}	23
non-blocking	←	channels	{"", "is buffered", "with buffer"}	8
multiple threads	←	multiple processors	{"", "use"}	19
multiple threads	←	concurrent programs	{"", use}	11
multiple threads	←	parallel programs	{"", "can use", "use"}	7
multiple threads	←	goroutines	{""}	7
multiple threads	←	multiple tasks	{"", "using", "on"}	5
multiple threads	→	multiple tasks	{""}	5
shared data	←	channels	{"", "contains", "enables"}	11
shared data	←	mutex	{"", "enables"}	5
shared data	←	concurrent programs	{""}	5
critical section	←	mutex	{"", "are used in a"}	8
typed values	←	channels	{"", "use"}	8

Table 4.7: Most frequent connections in the concurrency exercise

Tables 4.8 and 4.9 show the most frequent connections when it takes to concepts "non-blocking" and "multiple threads" respectively in the concurrency and parallelism exercise.

Connection			Labels	Occ.
non-blocking	←	goroutines	{""}	23
non-blocking	←	channels	{"", "with buffer", "is buffered"}	8

Table 4.8: Most frequent connections for "non-blocking" in the concurrency and parallelism exercise

Connection			Labels	Occ.
multiple threads	←	multiple processors	{"", "use"}	19
multiple threads	←	concurrent programs	{"", "use"}	11
multiple threads	←	parallel programs	{"", "can use", "use"}	7
multiple threads	←	goroutines	{""}	7

Table 4.9: Most frequent connections for "multiple threads" in the concurrency and parallelism exercise

Figures 4.6 and 4.7 show two examples of solutions for the concurrency and parallelism exercise provided by two of the research participants.

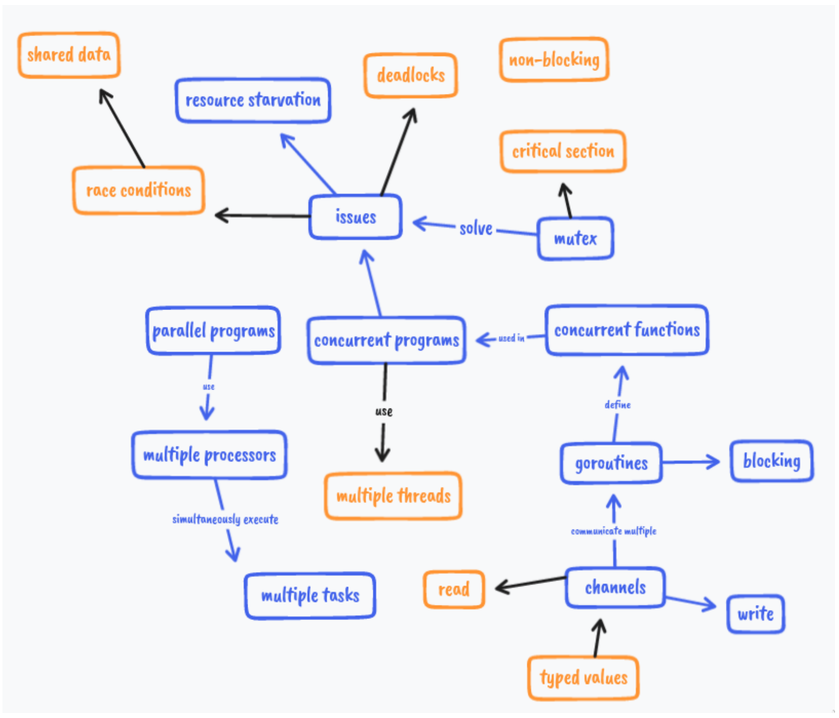


Figure 4.6: Concurrency and parallelism exercise solution (example 1)

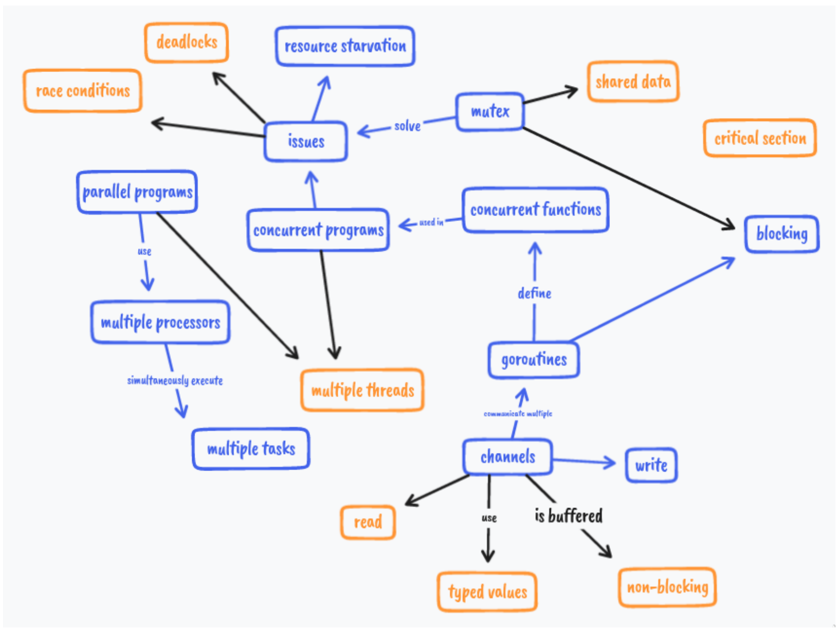


Figure 4.7: Concurrency and parallelism exercise solution (example 2)

Chapter 5

Discussion

In this chapter we discuss the main outcomes we have reached with the research and the approaches used when collecting and analyzing the data. First we comment on the results seen in the previous chapter, then we reflect on the used methodologies throughout the project execution.

5.1 Results

5.1.1 Programming background within participants

Looking at the plots from figure 4.1 we can get an idea on the overall programming experience from the research participants. We can also see which are the languages research participants are more comfortable with.

Even though almost half of the students (41.5%) just picked up programming this year, 7 students have been programming for more than 4 years and 24 of them in between 1 and 4. We can also see that most students had never used Go before the course but on the other hand most students had used Java and/or Python.

Analyzing the previously taken programming courses from the research participants, we were able to differentiate 9 student groups (check table 4.1). We can see how most research participants (81.13%) are within the first two groups. The groups from 3 to 9 are not significant as most of them consist of only one single participant.

5.1.2 Misconceptions about types in Go (RQ1)

As seen in table 4.2, most students (35 and 21 respectively) connected "basic types" to "strings" and "Go programming language" to "compiler based". This shows that most students got these two trivial connections right as Go is a compiler based language and the string type is grouped within the basic types of the language (together with numbers and booleans). However, not all students agree when it takes to less intuitive concepts.

If we take a look at table 4.3 we can see some students related "arrays" to "reference types", others related it to "aggregate types" and others to "basic types". In the Go programming language, arrays together with structs are aggregate types. This shows a misconception when it takes to arrays and data types.

Similar misconceptions are found looking at tables 4.4 and 4.5. In the case of "channels", 21 students related the concept to "reference types". This is a connection that makes sense, given that channels are one of the reference types that can be found in the language. However, 8 and 6 students related the concept to "aggregate types" and "basic types" respectively, which shows again a misconception when it takes to data types in Go.

Most participants related "functions" with "Go programming language" or "methods" instead of relating it to one of the types. Only 7 students related "functions" with "reference types".

In table 4.6 we see that the three most common connections when it takes to the "type inheritance" concepts are within the range of 8 and 5 occurrences. This shows that students connected the "type inheritance" concept in many different ways, also some of them might not have connected it at all. Overall we can see students are confused about the "type inheritance" concept. One of the main aspects of Go (and one of the main differences from Java) is that it does not feature type inheritance. However, students connected "Go programming language" to "type inheritance" and some of them used connection labels such as "has".

5.1.3 Perception of concurrency and parallelism (RQ2)

In table 4.7 we find a list with the most frequent connections in the concurrency and parallelism exercise. Again, we can see how most students connected the really basic concepts the same way. The most common connection is "channels" connected to "read", this is a predictable one given that in the exercise statement "write" was already connected to "channels". "issues" connected to "deadlocks" and "race conditions" are in second and third place with 23 and 24 connections respectively.

However, when taking a look at other concepts such as "multiple threads", we find a more variate set of connections. If we take a look at table 4.9, we see how most participants connected "multiple processors" to "multiple threads", some of them using the connection label "use". Participants also connected "multiple threads" from "concurrent programs" and "parallel programs" - using again connection labels such as "can use" or "use" - as well as "goroutines". All these connections make sense and could be considered valid. This shows how differently students can think about concepts and their relations.

5.1.4 Influence of prior programming knowledge (RQ3)

We tried to use data mining approaches and graph similarity algorithms in order to analyze the data. We were trying to cluster the participant takes based on different types of structures within the maps and see if the previous programming knowledge was somehow related to the results.

However, we could not get interesting or meaningful outcomes out of this part of the analysis. This types of techniques are really useful when the amount of data is very big [13]. Our final data-set being too small might be the reason we had to discard these methods in our analysis.

5.2 Method

5.2.1 CMTaks success

Developing CMTask took two weeks of work but it saved us a lot of time during the data collection.

The collected CMs were saved in JSON format by default. This helped us reduce big amounts of potential manual work related to CM transcription and encoding. After downloading all the CM files, parsing them from JSON to directed graphs was an intuitive and fast task.

Since the tool was being developed by us, we were able to fine-tune it as much as we felt necessary. We tried to make it as user-friendly as possible given the short amount of time we had. Some students noticed this and gave us positive feedback about the concept mapping exercises.

The website was up and running non-stop during the entire data collection process and all research participants were able to solve the CM exercises and answer the programming experience survey successfully with no technical issues at all.

5.2.2 Number of participants

We received data from a total of 71 research participants. Within these 71 participants there were 59 students and 12 TAs. We did not end up using the TA data as it was not significant enough. As far as the students, we had to remove data from 6 of them due to their CMs being vaguely defined. So we ended up using data from 53 students.

There were 290 students coursing DD1396. The 53 research participants are a 18.27% of all students taking the course. Getting data from more of them would have been useful in order to carry out a more data-driven analysis.

5.2.3 Data quality

Not all the CMs we received from research participants are equally detailed. While some of them were really detailed with abundant edge-labeling and additional concepts, there were also a considerable amount of vaguely-defined maps. Most students did not label edges (or labeled very few of them).

It is possible that different concept mapping familiarity levels among students has been the reason behind the quality gap within the data. Some students taking less time to work on the task than others can also be part of the problem.

Chapter 6

Conclusions

In this project we developed a web-based tool useful for the design and assessment of concept mapping exercises. We used the tool to carry out a concept map analysis on computer science student understanding of the Go programming language, concurrency and parallelism.

On the one hand we found some common student misconceptions when it takes to the Go programming language data-types. On the other hand we saw how differently students relate concepts such as threads and processors within the field of concurrency and parallelism. We were not able to find a meaningful relationship between student programming background and their concept mapping exercise takes.

From the concept mapping tooling perspective, future work after this project could be expanding the development of CMTask into a production ready platform with additional features where everyone could design and assess their own concept mapping exercises. Instead of just gathering the concept maps as JSON, it would be interesting to provide data analysis features within the same platform.

From the experiment perspective, it would be interesting to evaluate it within a bigger crowd of participants. This could lead to interesting results when analyzing the programming experience influence. Graph similarity and subgraph matching could be properly executed and potentially give meaningful insights.

Bibliography

[1] Malini Mistry. “Computer science education: perspectives on teaching and learning in school”. eng. In: *Education 3-13* 48.2 (2020), pp. 253–254.

[2] Matti Tedre, Simon, and Lauri Malmi. “Changing aims of computing education: a historical survey”. eng. In: *Computer Science Education* 28.2 (2018), pp. 158–186.

[3] Nasser Giacaman and Joel Adams. “Introductory Concurrency and Parallelism Education”. In: *Proceedings of the ACM Conference on Global Computing Education*. CompEd ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 171. ISBN: 9781450362597.

[4] Association for Computing Machinery (ACM) Joint Task Force on Computing Curricula and IEEE Computer Society. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. Association for Computing Machinery, 2013. ISBN: 9781450323093.

[5] Dario M Torre, Steven J Durning, and Barbara J Daley. “Concept Maps: Definition, Structure, and Scoring”. eng. In: *Academic medicine* 92.12 (2017), pp. 1802–1802. ISSN: 1040-2446.

[6] Vinicius dos Santos et al. “Analyzing the Use of Concept Maps in Computer Science: A Systematic Mapping Study”. eng. In: *Informatics in Education* 16.2 (2017), pp. 257–288. ISSN: 1648-5831.

[7] Alan A. A Donovan and Brian W Kernighan. *The Go Programming Language*. eng. Addison-Wesley Professional Computing Series. Addison-Wesley Professional, 2015. ISBN: 9780134190570.

[8] Barbara J Daley et al. “Integrative Literature Review: Concept Mapping: A Strategy to Support the Development of Practice, Research, and Theory Within Human Resource Development”. eng. In: *Human Resource Development Review* 9.4 (2010), pp. 357–384. ISSN: 1534-4843.

[9] Ian M. Kinchin. “Concept mapping in biology”. eng. In: *Journal of biological education* 34.2 (2000), pp. 61–68. ISSN: 0021-9266.

[10] I. M Kinchin and L. B Cabot. “An introduction to concept mapping in dental education: the case of partial denture design”. eng. In: *European journal of dental education* 13.1 (2009), pp. 20–27. ISSN: 1396-5883.

[11] Jeroen Keppens and David Hay. “Concept map assessment for teaching computer programming”. eng. In: *Computer science education* 18.1 (2008), pp. 31–42. ISSN: 0899-3408.

[12] Kate Sanders et al. “Student understanding of object-oriented programming as expressed in concept maps”. eng. In: *Proceedings of the 39th SIGCSE technical symposium on computer science education*. SIGCSE ’08. ACM, 2008, pp. 332–336. ISBN: 1595937994.

[13] Andreas Mühling. “Concept Landscapes: Aggregating Concept Maps for Analysis”. eng. In: *Journal of educational data mining* 9.2 (2017), p. 1. ISSN: 2157-2100.

TRITA – XXX-XXX 20XX:XX