Approach to Develop a Concept Inventory Informing Teachers of Novice Programmers' Mental Models

Henry Julie
Namur Digital Institute
University of Namur, Belgium
julie.henry@unamur.be

Dumas Bruno
Namur Digital Institute
University of Namur, Belgium
bruno.dumas@unamur.be

Abstract—This Research Full Paper proposes an approach for developing and administering a concept inventory (CI). Based on misconceptions, the CI should allow the identification of mental models, but more importantly, the visualization of the evolution of these mental models - from intuitive to correct mental models - over a given period of teaching time. This approach is illustrated by the development of three questionnaires, each focusing on a basic programming concept: Variables, conditional statements, and functions. Its main steps are: Identifying topics and students' misconceptions, creating questions that reflect these misconceptions, administering questions to students, and selecting most relevant questions to become part of the CI. The research was conducted over four years with four groups of students, as part of an introductory programming course. During the first year, an ethnographic approach was applied to define the problematic topics and identify the misconceptions of the students. From the second to the fourth year, each year and for one semester, students were assessed longitudinally, according to a specific schedule. The developed questionnaires were iterated three times and administered to more than 250 students.

Index Terms—Methodology, Questionnaire, Misconceptions, Misunderstanding, Perceptions, Assessment, CS1

I. INTRODUCTION

Introductory programming courses result in rather high failure and dropout rates. Amongst other factors, it appears that novice programmers often hold misconceptions of programming constructs that hinder their progress and may discourage them from continuing their learning. Students use misconceptions, inter alia, to build their incorrect mental models. Identifying misconceptions and mental models should help teachers to provide appropriate intervention in the future. Moreover, misconceptions are a prime source of material to build assessment instruments, as concept inventories.

A concept inventory (CI) is a multiple-choice questionnaire that seeks to measure a student's knowledge of a set of concepts and helps reveal their thinking patterns. In general, a CI is administered before and after a period of teaching. This is not the approach chosen in computer science education, as students supposedly have no preconception of the field. Teachers have to help students to build viable mental models at an early stage. If they don't, students may build inappropriate mental models based on the misuse of their prior knowledge and intuitive mental models. So, the understanding of students' intuitive mental models is crucial for preparing suitable learning materials.

The research presented here proposes an approach for developing and administering a CI based on misconceptions, allowing the identification of mental models, but more importantly, allowing the visualization of the evolution of these mental models - from intuitive to correct mental models - over a given period of teaching time. This approach is illustrated by the development of three questionnaires, each focusing on a basic programming concept: Variables, conditional statements, and functions.

II. RELATED WORK

A. Effective Mental Models

According to Johnson-Laird [1], mental models are incomplete, dynamic, and working models that humans use to understand the world. In an educational context, they are models that students create from a physical phenomenon, the purpose of which is to be useful in enabling them to understand these phenomenons [2]. They are built from the students' perception of the world, or the comprehension of discourse.

Ben-Ari reveals that *"a (beginning) computer science student has no effective (mental) model of a computer"* [3]. *"Since computer science deals with artifacts—programming languages and software, the creator of the artifact employed a very detailed model and the learner must construct a similar, though not necessarily identical, model"* [3]. Norman [4] suggests that a user constructs the mental models of computer systems (*target system*) through interacting with them and constantly refines the models throughout these interactions. According to Ben-Ari [3], the models of computer artifacts have to be explicitly taught. Students often take a programming course without the effective mental models they need to develop an appropriate understanding of the learning material. Students are not able to build mental models just by listening to lectures or reading textbooks. If they do, they may build inappropriate mental models based on the misuse of their prior knowledge and intuitive models.

Gentner [5] explained that the understanding of students' intuitive mental models is crucial for preparing suitable learning materials: *"If typical incorrect models are understood, then instructors and designers can create materials that minimize the chances of triggering errors"*. Teachers should explicitly help students to build viable mental models at an early stage so that students themselves do not build incorrect mental models.

In the context of computer science education, some studies have been conducted to investigate students' mental models of sequence and assignment [6], [7], object, variable and rule [8], I/O and control flow [9], and recursion [10], [11]. Each of these studies identified students' mental models from closed-ended questionnaires or interviews. Some authors [12]–[15] elicited, with closed-ended questionnaires as well as interviews, the mental models that novice programmers have of fundamental programming concepts.

However, the existing work is generally limited to a one-time measurement of mental models at the end of learning. Therefore, the research presented here also focuses on intuitive mental models. If students have to build viable mental models at an early stage, the teacher has to help them during the first courses, especially because these courses are often the most attended.

B. Existing Conceptions and Misconceptions

It is not easy for students to discard their existing conceptions and adopt new ones [16]. Students often cannot realize that their existing conceptions are in conflict with the taught conceptions [17]. It is important, therefore, that teaching is able to help students to become aware of this issue and then to help them to construct coherent conceptions. Existing conceptions are sometimes misconceptions. In the context of programming education, Sorva [18] defined misconceptions as *"understandings that are deficient or inadequate for many practical programming contexts"*, including among others, misunderstanding of concepts. Quian [19] identified misconceptions as errors in conceptual understanding, misunderstanding of programming constructs such as variables and assignment statements, conditional expressions or loops. These misconceptions have to be identified to allow teachers to provide appropriate intervention in the future [20]. In addition to assessing students' understanding of the core concepts in introductory programming, it is recommended to explore the evolution of misconceptions in relation to strategies and tools used in courses, but also to individual differences [19]. Factors usually linked with misconceptions include incomplete or non-viable mental models [19].

According to Maier [21], the way to resolve or prevent misconceptions is to directly confront the learner with an experience that causes an imbalance. Challenging students' existing ideas encourages them to detect problems in their understanding and to motivate them to build appropriate understandings [22]. Generally, a cognitive conflict teaching strategy involves three steps: Investigating students' prior knowledge and existing conceptions; challenging students with contradictory information; evaluating the conceptual change between students' prior ideas and current ones [23].

However, the existing work that focuses on misconceptions generally does not refer to mental models. Yet the two notions are related: Students use misconceptions, inter alia, to build their incorrect mental models. Because students have to be confronted with their misconceptions, these are a prime source of material for teachers to build, in a first step, assessment instruments. These instruments allow, in a second step, to identify students' mental models.

C. Concept Inventory

According to Wittie et al., a concept inventory (CI) is *"a research-based multiple-choice test that seeks to measure a student's knowledge of a set of concepts while also capturing conceptions and misconceptions they may have about the topic under consideration"* [24]. Developing a CI assessing students' understanding of the core concepts in introductory programming allows teachers to identify students' misconceptions and then to provide appropriate intervention in the future [20]. This assessment instrument can be used as a diagnostic test, to identify appropriate teaching and learning activities, to assess the impact of changing instruction methods on students' understanding, to give feedback to students, to compare instructional methods, or to evaluate overall learning and instructional effects.

In computer science education, CIs have been proposed [20], among others, for binary search trees [25], [26], digital logic [27], hash tables [26], operating systems [28], sequence and assignment [6], and CS1 programming fundamentals [29], [30].

Adams and Wieman describe an established procedure for developing and validating a CI, the last three steps of which are iterative [31].

1) Establish topics by reading domain programming education literature, self-reflection, and discussion with experienced teachers or subject experts.
2) Identify how students' thinking deviates from expert thinking by observing and interviewing them, consulting experienced teachers and domain misconception literature.
3) Create open-ended questions and administer them to students to further examine issues raised in the previous step.
4) Create closed-ended questions and establish distractors based on the above steps.
5) Validate questions by reaching consensus among experts that all responses are correct, ensuring that students interpret the questions consistently, and that maladjusted student thinking results in incorrect responses.
6) Administer the CI to large populations, applying statistics to account for reliability and validity.

In contexts other than computer science education, a CI is administered before (pre-test) and after (post-test) a period of teaching. However, it has been argued that computer science
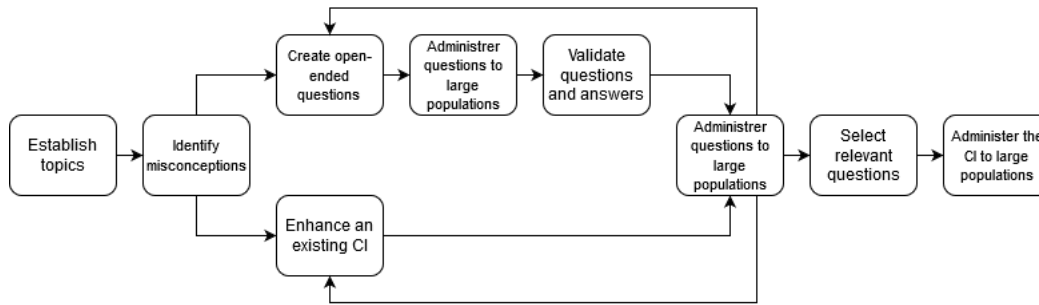
Fig. 1: CI development approach

students do not have a preconception of the field and, therefore, that administering a CI as a pre-test may not be useful: *"It is likely that a majority of student misconceptions are a result of instruction in computer science rather than based upon a set of common, naive understandings [that students] bring to the topic from their experience in the world"* [30]. Currently, computer science topics are increasingly part of the K-12 curriculum. Because students will more and more have discovered basic programming concepts before the university, the CI as a pre-test is becoming more and more relevant.

The research presented here proposes an approach for developing and administering a CI based on misconceptions, allowing the identification of mental models, but more importantly, allowing the visualization of the evolution of these mental models - from intuitive to correct mental models - over a given period of teaching time.

III. METHODOLOGY

An introductory programming course organized in the first year of a bachelor's degree at the University of Namur is the measurement ground for the research described in this article. This is a mandatory course for both students with a Computer Science major and those with a Business Engineering one. More than 120 students enrol each year and attend four 60-minute lectures (theoretical classes) and three 60-minute labs (practical classes) every week over a 13-week semester. Each week, a new core programming concept (variable, conditional statement, function, loop, among others) is covered. These concepts are exemplified in Python.

A CI is developed for the course following an approach similar to Adams and Wieman [31] which is depicted in Figure 1. This development takes place over 4 years. The main steps are: Identify topics and students' misconceptions, create questions that reflect these misconceptions, administer questions to students, and select most relevant questions to become part of the CI.

A. Identify Topics and Related Misconceptions

During one semester in year 1, an ethnographic approach was applied to define problem topics and to identify misconceptions of students taking the introductory programming course. This approach was repeated in the second semester of the same year, with the same students, as part of a large-scale

programming project (+/- 10,000 lines of code) that students have to undertake by applying what they have learned in the introductory programming course.

To identify misconceptions, students were observed individually and in groups while programming. Informal discussions were held regularly with them. Reading domain misconception literature has helped to improve the observations. To define problem topics, discussions were held each week over the semester with the four teachers in charge. The results of the exams were reviewed with these teachers. Informal discussions were also held with experts in the field, teachers in other contexts.

On the basis of the results obtained, it was decided that the CI would focus on the first three concepts covered in the course, namely the **variable**, the **if statement**, and the **function**. These concepts correspond to the concepts learned at the time of the dropout peak among students following the introductory programming course.

B. Create Questions Reflecting Misconceptions

The question creation step varies depending on the programming concept under consideration. One approach is based on existing work, another is based on the observations made during the ethnographic study.

Because the identification of mental models is crucial in this research, a published questionnaire measuring the mental models used by the students to solve **variable** assignment problems [6] forms the basis for the **questionnaire on variables**. The questions that it contains are also inspiration for the creation of questions in the two questionnaires on **if statements** and on **functions**. Ethnographic observations highlighted the major difficulties that students encounter with the **if statement** and **function programming concept**. On the basis of these results, open-ended questions (inspired by the questionnaire on variables) were created to collect data to define the mental models used by students to solve if statement and function problems.

For all three questionnaires, three iterations were conducted (starting from year 2) with each time a different group of students from the two populations described, i.e. computer science and business engineering students.
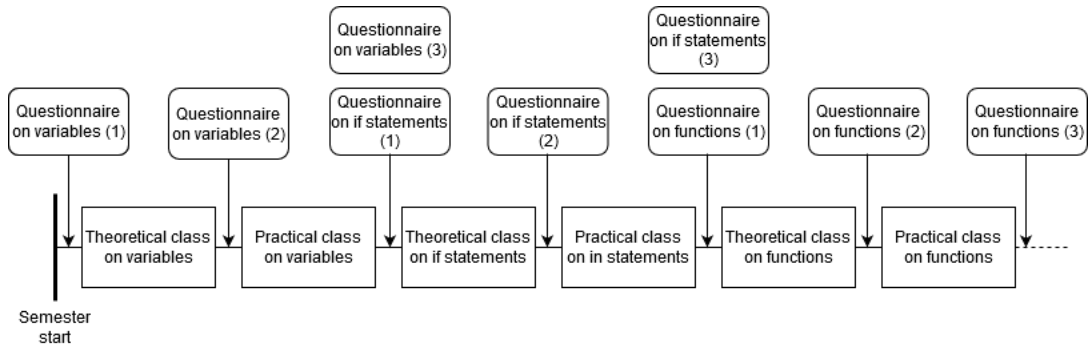
Fig. 2: Organization, on a timeline, of questionnaires administration during one semester

C. Administer Questions to Students

Each questionnaire was administered three times each year from year 2 to year 4. Data were collected at specific times throughout a semester (Figure 2), inspired by pretest-post-test design [32]. For each concept, the questionnaire was administered before the pertaining theoretical class, between the theoretical class and the pertaining practical class, and after the practical class. The questionnaires were presented to the students as a formative assessment [33]. It was not specified to the students that the questionnaires were the same for all three administrations. If some students were still giving wrong answers after a third questionnaire on the same topic, they were contacted individually and offered help.

D. Select Relevant Questions

This step is, according to the approach presented here, the last hurdle to develop the CI. At the time of writing, analyses of the results allowing for a justified selection of questions are underway. However, some selection criteria can already be put forward. Three concepts will be tested in the CI. The number of questions per concept will have to be defined according to the number of different mental models used by the students, and the number of misconceptions identified for each concept. It will be necessary to measure the weight to be given to the different types of questions in a questionnaire (e.g. single, double, and triple assignments in the questionnaire on variables). Finally, it will also be necessary to measure how useful the repetition of certain types of questions is.

IV. Questionnaire on Variables

A. Development

Dehnadi's questionnaire[1] focuses on the variable program-ming concept and on associated mental models [6]. It was the subject of several scientific papers [34]–[37] and has been selected as it allows for comparisons with other works.

Dehnadi's questionnaire contains 12 elements and presents three types of assignment problems to solve, exemplified in Figure 3: Single (3 questions), double (3) and triple two-variable assignments (6). The problems are presented in the

[1]Bornat proposed in 2016 an online version of this questionnaire - http://www.eis.mdx.ac.uk/staffpages/r_bornat/tests/distribution.html

form of multiple-choice variable assignment problems, where respondents have to choose the correct values contained in the variables from a set of proposed answers (Figure 4).

The problems are proposed in the Java language. Students learn programming concepts in a theoretical way. The lan-guage used in the introductory programming course, Python, is only one way to implement them. It is expected that the students will be able to transfer their knowledge to another language, especially Java. The differences in syntax between an assignment in Python and an assignment in Java are minor: Typing at variable declaration and semicolons. The students did not even raise these differences during the administration of the questionnaire.

```
                                          int a = 5;
                                          int b = 3;
                       int a = 10;        int c = 7;
  int a = 10;          int b = 20;
  int b = 20;                             b = a;
                       b = a;             c = b;
  a = b;               a = b;             a = c;

  (a) Single          (b) Double         (c) Triple
```
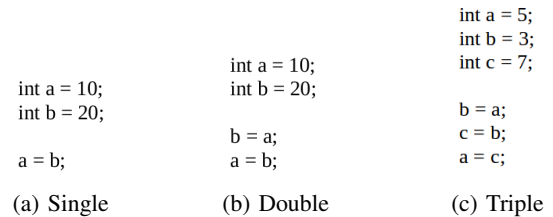
Fig. 3: Examples of assignment problems

Dehnadi identified eleven different mental models (Table I) used by the students to solve the proposed assignments. Each multiple-choice question includes a unique correct answer and distractors (Figure 4) that are carefully constructed to address all Dehnadi's mental models. A last open choice ("other") is always present if the student wishes to propose a new solution. Dehnadi's questions can be related to the major misconceptions of variables identified by Sorva [18] (Table II, Figure 4).

In year 3, an enhanced version of the Dehnadi's question-naire was created. Three questions were added, increasing the number of questions to 15. These are duplicates, formatted in a block programming language, of existing questions (Figure 5): A single, a double, and a triple assignment problems. These questions have been added to identify whether the answers are dependent on the language. They have been strategically placed in the questionnaire so that they are not placed directly after their Java counterpart.

1.
int a = 10;
int b = 20;
a = b;
	a = 20, b = 0	M1	MIS11

	a = 20, b = 20	M2 or M11	MIS9a
	a = 0, b = 10	M3	MIS11
	a = 10, b = 10	M4 or M11	MIS9a, MIS11
	a = 30, b = 0	M5	MIS9b
	a = 30, b = 20	M6	MIS9b
	a = 0, b = 30	M7	MIS9b, MIS11
	a = 10, b = 30	M8	MIS9b, MIS11
	a = 20, b = 10	M9	MIS12
	a = 10, b = 20	M10	MIS9b

Fig. 4: A sample question from the original Dehnadi's questionnaire

3′. set a to 10 set b to 20 set a to b	a = 20, b = 0	M1	MIS11
	a = 20, b = 20	M2 or M11	MIS9a
	a = 0, b = 10	M3	MIS11
	a = 10, b = 10	M4 or M11	MIS9a, MIS11
	a = 30, b = 0	M5	MIS9b
	a = 30, b = 20	M6	MIS9b
	a = 0, b = 30	M7	MIS9b, MIS11
	a = 10, b = 30	M8	MIS9b, MIS11
	a = 20, b = 10	M9	MIS12
	a = 10, b = 20	M10	MIS9b

Fig. 5: An assignment problem in block programming language

In year 4, the questionnaire was composed of 16 questions. One question was duplicated and enriched with a metaphorical representation of variables: The box metaphor. The idea was to measure the impact of a visual aid to diminish the abstract character of the variables.
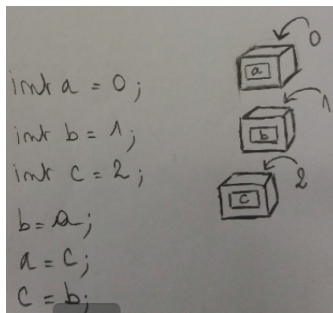

Fig. 6: An assignment problem enriched with the box metaphor

B. Administration

In year 2, Dehnadi's questionnaire, in its original version, was administered to 107 students [38]. This questionnaire was expanded by 3 questions (15-elements questionnaire) and was administered to 112 students in year 3 [39]. Finally, in year 4, 104 students completed the 16-elements questionnaire. Over three years of research, data was collected from 323 students through nine questionnaires (three years, with three administrations yearly).

C. Concluding Comments

Inspired by Dehnadi's questionnaire [6] and enhanced to measure intuitive mental models, the 16-element questionnaire on variables has proven its effectiveness. No mental model

has been added to the list defined by Dehnadi (Table I) . The results obtained during the three-year administration period make it possible to envisage, as future work, the phase of selection of the relevant questions by measuring the impact of question redundancy and attempts to diversify the way problems are written (blocks programming language, logigram, metaphors, etc.).

V. QUESTIONNAIRE ON IF STATEMENTS

A. Development

Based on the results of the ethnographic approach conducted in the first year, the if statement seems to be the most easily understandable programming concept for students. Consequently, a limited number of questions were asked in the first version of the questionnaire devoted to this concept. These questions are small if statement problems, comparable to the Dehnadi's questionnaire. Students have to resolve which values are stored in "a" and "b" variables after code execution.

In year 2, the questionnaire on if statements presents only three problems to solve, exemplified in Figure 7: If statement consisting of a "then" clause and a condition that is true (a), if statement consisting of "then" and "else" clauses and a condition that is false (b), and nested if statements and two conditions that are false (c). The problems were presented in the form of open-ended questions. Because of the minor differences in syntax between an if statement in the Python language and a if statement in the Java language, the problems are proposed in Java, as in the questionnaire on variables. Based on the results of the administration and informal discussions with students about the questionnaire, seven mental models were identified (Table III).

(a) If statement, "then" clause

(b) If statement, "then-else" clauses

(c) Nested if statements

Fig. 7: If statement problems included into the first version of the questionnaire

In year 3, one question in the Java language was added: An if statement consisting of a "then" clause and a condition that is false. Two questions (one if-then and one if-then-else statements) were duplicated and formatted in a block programming language. The questionnaire was then composed of 6 questions. Each question is a multiple-choice question

TABLE I: Dehnadi's mental models for the assignment of variables [6]. *Please note that the numbering does not correspond to the one proposed by Dehnadi.*

Mental Model	Description
M1	The value of the variable on the right is assigned to the variable on the left; the variable on the right is initialized to 0
M2	The value of the variable on the right is assigned to the variable on the left; the variable on the right retains its original value
M3	The value of the variable on the left is assigned to the variable on the right; the variable on the left is initialized to 0
M4	The value of the variable on the left is assigned to the variable on the right; the variable on the left retains its original value
M5	The value of the variable on the right is added to the value of the variable on the left; the variable on the right is initialized to 0
M6	The value of the variable on the right is added to the value of the variable on the left; the variable on the right retains its original value
M7	The value of the variable on the left is added to the value of the variable on the right; the variable on the left is initialized to 0
M8	The value of the variable on the left is added to the value of the variable on the right; the variable on the left retains its original value
M9	The values assigned to the two variables are exchanged
M10	Variables retain their original values
M11	Mathematical equality is applied

TABLE II: Sorva's misconceptions related to the variable programming concept [18]

ID	Description
MIS8	Magical parallelism: Several lines of a (simple non-concurrent) program can be simultaneously active or known
MIS9a	A variable can hold multiple values at a time
MIS9b	A variable "remembers" old values
MIS10	Variables always receive a particular default value upon creation
MIS11	Primitive assignment works in opposite direction
MIS12	Primitive assignment works both directions (swaps)

including a unique correct answer and distractors (Figure 8) that are carefully constructed to address all identified mental models. A last open choice ("other") is always present if the student wishes to propose a new solution. Some questions can be related to the major misconceptions of if statements identified by Sorva [18] (Table IV, Figure 8). However it appears that, on the one hand, some misconceptions remain difficult to measure through a question, and on the other hand, that some answers provided by students reveal misconceptions not identified by Sorva.

1.
int a = 5;
int b = 3;
if (a > b) {
    a = a + b;
}
b = a;
	a = 5, b = 5	M1	MIS23

	a = 8, b = 8	M2 or M6	/ or MIS26, MIS28, MIS36
	a = 8, b = 5	M3	
	a = 8, b = 3	M4 or M7	MIS24, MIS26
	a = 5, b = 3	M5	

Fig. 8: A sample question from the questionnaire on if statements

In year 4, two problems were duplicated and proposed in the form of a logigram (Figure 9). The idea was to measure the impact of a visual aid to diminish the abstract character of the if statement. The number of questions was thus increased to 8.

B. Administration

In year 2, the questionnaire on if statements, in its first version, was administered to 112 students. This questionnaire was expanded by 3 questions (6-elements questionnaire) and was administered to 101 students in year 3. Finally, in year
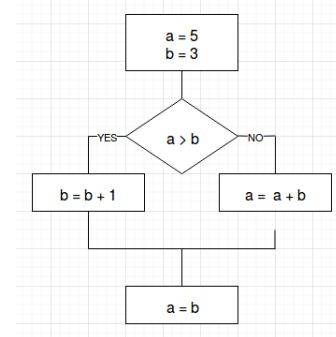

Fig. 9: If statement problem in the form of a logigram

4, 108 students completed the 8-elements questionnaire. Over three years of research, data was collected from 321 students through nine questionnaires (three years, with three questionnaires yearly).

C. Concluding Comments

The different questionnaires on if statements highlighted seven mental models (Table III). Because this concept is more easily acquired by students, the list of mental models seems to be exhaustive. The next step is to select the relevant questions. Even if the number of questions (8) currently being tested is small, taking them all is not necessarily useful.

VI. QUESTIONNAIRE ON FUNCTIONS

A. Development

The results presented in this section are based on the same number of iterations as the previous questionnaires, i.e.

TABLE III: Mental models for the if statements

Mental Model	Description
M1	*If statement* is not executed, nor "then" (even if the condition is true) nor "else" clause (if it exists). Code before and after *if statement* is correctly executed.
M2	Code before and after, and *if statement* ("then" **OR** "else" - if it exists - clause according the condition) are correctly executed.
M3	Code executed in *if statement* has an effect only on the variables that are changed in it. Code after *if statement* is executed but is not affected by the changes made in *if statement*.
M4	Code before and *if statement* ("then" **OR** "else" - if it exists - clause according the condition) are correctly executed. Code after *if statement* is not executed.
M5	*If statement* is not executed, nor "then" (even if the condition is true) nor "else" clause (if it exists). Code after is considered as part of *if statement* and then is not executed.
M6	Code before and after, and if statement ("then" **AND** "else" - if it exists - clauses regardless of condition) are executed.
M7	Code before and *if statement* ("then" **AND** "else" - if it exists - clauses regardless of condition) are executed. Code after *if statement* is not executed.

TABLE IV: Sorva's misconceptions related to the conditional statement programming concept [18]

ID	Description
MIS23	Difficulties in understanding the sequentiality of statements
MIS24	Code after if statement is not executed if the then clause is
MIS25	If statement gets executed as soon as its condition becomes true
MIS26	A false condition ends program if no else branch
MIS27	Both then and else branches are executed
MIS28	The then branch is always executed
MIS29	Using else is optional (the next statement is always the else branch)
MIS36	All statements of a program get executed at least once

three iterations. It appears that this number of iterations is not sufficient to identify all the mental models used by the students. The situation proves to be much more complex than for the other concepts. Moreover, according to the observations made during ethnographic approach, students have difficulty to express their misconceptions about the function. From a literature perspective, the misconceptions defined by Sorva cover more than the paradigm discussed in the introductory programming course which is the specific context of this research. Thus it was decided, as a first approach, to focus on the most prevalent misconceptions to build questions.

In its first version (year 2), the questionnaire on functions presents only four problems to solve, exemplified in Figure 10: Code with defined functions but no function is called (a) - two questions -, code with defined functions, a function is called but return value is not stored (b), and code with defined functions, functions are called, and return value is stored (c). The problems were presented in the form of open-ended questions.

Because of the differences in syntax between a function declaration in the Python language and a function declaration in the Java language, the problems are proposed in Python. Indeed, given the difficulties encountered by the students with the concept of function, the change of language would have added problems related more to syntax than to the understanding of the concept.

In year 3, three additional questions expanded the questionnaire (Figure 11). These questions focus on misconceptions about function parameters: Two calls of a same function, but with different parameters (a), a call with numeric parame-
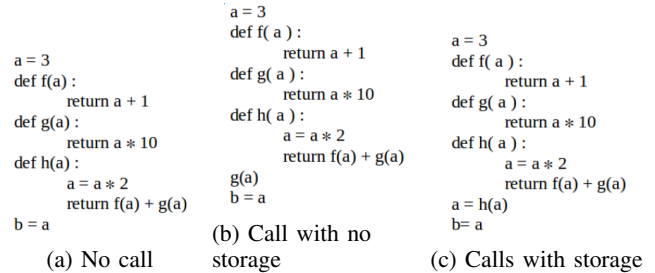

Fig. 10: Function problems included in the first version of the questionnaire

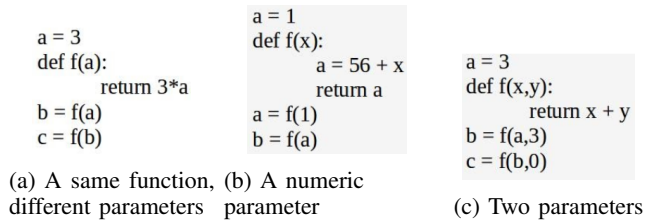ter (b), and calls of functions with two parameters (c).

Fig. 11: Function problems added in year 3

Given the wide variety of responses provided in years 2 and 3, it was difficult to define the distractors of the multiple-choice questions. The most frequent answers were proposed as distractors in year 4. An attempt was made to define the mental models associated with these responses, but these definitions still need to be improved and the list of mental models is

TABLE V: Mental models for the functions (question 4)

Mental Model	Description
M1	*Function* code is perhaps executed but no value is returned. "What happens in the function stays in the function". Code before and after the call is executed.
M2	Code before the call, called *functions* codes and code after the call are correctly executed.
M3	*Function* code is partially executed. *Function* calls within a *function* are not taken into account. The "return" keyword causes by default the return of the variable passed in parameter. Code before and after the call is executed.
M4	Code before the call, called *functions* codes and code after the call are executed, but the value of the variable passed in parameter remains constant despite the *function* codes.
M5	All Code is executed according to the order in which the *functions* are defined
M6	*Function* code is partially executed: Only the "return" instructions are executed. Code before and after the call is executed.

TABLE VI: Selected Sorva's misconceptions related to the function programming concept [18]

ID	Description
MIS37	Subprogram code is executed according to the order in which the subprograms are defined
MIS38	A return values does not need to be stored (even if one needs it later)
MIS39	A method can be invoked only once
MIS40	Numbers or numeric constants are the only appropriate actual parameters corresponding to integer formal parameters
MIS42	Difficulties understanding the invocation of a method from another method
MIS43	Confusion over where return values go
MIS46	Parameter passing requires different variable names in call and signature
MIS47	Subprograms can (routinely) use the variables of calling subprograms
MIS50	A function (always) changes its input variable to become the output
MIS53	Upon return, the value of a variable changes to match a previously given parameter

4.	b = 3	M1	MIS49
a = 3	b = 67	M2	/
def f(a):	b = 6	M3	MIS38, MIS42, MIS53
return a + 1	b = 40	M4	MIS47
def g(a):	b = 80	M5	MIS37
return a * 10	b = 34	M6	/
def h(a):			
a = a * 2			
return f(a) + g(a)			
a = f(a)			
b = a			

Fig. 12: A sample question from the questionnaire on functions

incomplete. Identification work shall also be continued with respect to misconceptions related to each question (Table VI, Figure 12).

B. Administration

In year 2, the questionnaire on functions, in its first version, was administered to 111 students. This questionnaire was expanded by 3 questions and was administered to 94 students in year 3, and to 57 students in year 4. Over three years of research, data were collected from 262 students through nine questionnaires (three years, with three administrations yearly).

C. Concluding Comments

The questionnaire on functions is still under development at the time of writing this article. It appears that this number of iterations is not sufficient to identify all the mental models used by the students. The situation for the function programming concept proves to be much more complex than for the other concepts. The step of selecting the relevant questions to create the CI is, therefore, not yet possible for the function concept.

VII. CONCLUSION AND FUTURE WORK

This research proposes an approach for developing and administering a concept inventory (CI). Based on misconceptions, the CI should allow the identification of mental models, but more importantly, the visualization of the evolution of these mental models - from intuitive to correct mental models - over a given period of teaching time. The approach is illustrated by the development of three questionnaires, each focusing on a basic programming concept: Variables, conditional statements, and functions. Its main steps are: Identifying topics and students' misconceptions, creating questions that reflect these misconceptions, administering questions to students, and selecting most relevant questions to become part of the CI. The research was conducted over four years with four groups of students (one different per year), as part of an introductory programming course. During the first year, an ethnographic approach was applied to define the problematic topics and identify the misconceptions of the students (group 1) taking the course. From the second to the fourth year, each year and for one semester, students (from group 2 to group 4) were assessed longitudinally, according to a specific schedule. The developed questionnaires were iterated three times and administered to more than 250 students.

All questionnaires appear to be effective in identifying mental models. Therefore, the CI composed of a selection of questions from each of these questionnaires should also be effective in this task. Although the results were not processed for all questionnaires, the studies conducted on the questionnaire on variable concerning the evolution of mental models show that the administration methodology is effective.

It will then be interesting to implement the CI in different contexts (by adapting the administration methodology) and to measure its interest and impact as a pedagogical tool (helping to detect difficulties among novices, assessing the knowledge of novices, etc.).

REFERENCES

[1] P. N. Johnson-Laird, *Mental Models: Towards a Cognitive Science of Language, Inference, and Consciousness*. Harvard University Press, 1983, no. 6.

[2] I. M. Greca and M. A. Moreira, "Mental models, conceptual models, and modelling," *International journal of science education*, vol. 22, no. 1, pp. 1–11, 2000.

[3] M. Ben-Ari, "Constructivism in computer science education," *Journal of Computers in Mathematics and Science Teaching*, vol. 20, no. 1, pp. 45–73, 2001.

[4] D. A. Norman, "Some observations on mental models." *Lawrence Erlbaum:*, p. 99, 1983.

[5] D. Gentner and A. L. Stevens, *Mental Models*. Psychology Press, 2014.

[6] S. Dehnadi, "A cognitive study of learning to program in introductory programming courses." Ph.D. dissertation, Middlesex University, 2009.

[7] R. Bornat, S. Dehnadi *et al.*, "Mental models, consistency and programming aptitude," in *Proceedings of the tenth conference on Australasian computing education-Volume 78*. Australian Computer Society, Inc., 2008, pp. 53–61.

[8] M. A. Sasse, "Eliciting and describing users' models of computer systems," Ph.D. dissertation, University of Birmingham, 1997.

[9] P. Bayman and R. E. Mayer, "A diagnosis of beginning programmers' misconceptions of basic programming statements," *Communications of the ACM*, vol. 26, no. 9, pp. 677–679, 1983.

[10] H. Kahney, "What do novice programmers know about recursion," in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. ACM, 1983, pp. 235–239.

[11] D. M. Kurland and R. D. Pea, "Children's mental models of recursive logo programs," *Journal of Educational Computing Research*, vol. 1, no. 2, pp. 235–243, 1985.

[12] V. Vainio and J. Sajaniemi, "Factors in novice programmers' poor tracing skills," in *ACM SIGCSE Bulletin*, vol. 39, no. 3. ACM, 2007, pp. 236–240.

[13] L. Ma, J. Ferguson, M. Roper, and M. Wood, "Investigating and improving the models of programming concepts held by novice programmers," *Computer Science Education*, vol. 21, no. 1, pp. 57–80, 2011.

[14] L. Ma, "Investigating and improving novice programmers' mental models of programming concepts," Ph.D. dissertation, University of Strathclyde, 2007.

[15] J. Sajaniemi, M. Kuittinen, and T. Tikansalo, "A study of the development of students' visualizations of program state during an elementary object-oriented programming course," *Journal on Educational Resources in Computing (JERIC)*, vol. 7, no. 4, p. 3, 2008.

[16] J. Davis, "Conceptual change," *Emerging perspectives on learning, teaching, and technology*, vol. 19, 2001.

[17] P. W. Hewson and M. G. Hewson, "The role of conceptual conflict in conceptual change and the design of science instruction," *Instructional Science*, vol. 13, no. 1, pp. 1–13, 1984.

[18] J. Sorva, V. Karavirta, and L. Malmi, "A review of generic program visualization systems for introductory programming education," *ACM Transactions on Computing Education (TOCE)*, vol. 13, no. 4, p. 15, 2013.

[19] Y. Qian and J. Lehman, "Students' misconceptions and other difficulties in introductory programming: A literature review," *ACM Trans. Comput. Educ.*, vol. 18, no. 1, pp. 1:1–1:24, Oct. 2017.

[20] C. Taylor, D. Zingaro, L. Porter, K. C. Webb, C. B. Lee, and M. Clancy, "Computer science concept inventories: past and future," *Computer Science Education*, vol. 24, no. 4, pp. 253–276, 2014.

[21] S. Maier, "Misconception research and piagetian models of intelligence," in *Proc. 2004 Oklahoma Higher Education Teaching and Learning Conference*, 2004.

[22] P. Scott, H. Asoko, and R. Driver, "Teaching for conceptual change: A review of strategies," *Research in physics learning: Theoretical issues and empirical studies*, pp. 310–329, 1992.

[23] M. Limón, "On the cognitive conflict as an instructional strategy for conceptual change: A critical appraisal," *Learning and instruction*, vol. 11, no. 4-5, pp. 357–380, 2001.

[24] L. Wittie, A. Kurdia, and M. Huggard, "Developing a concept inventory for computer science 2," in *Frontiers in Education Conference (FIE)*. IEEE, 2017, pp. 1–4.

[25] H. Danielsiek, W. Paul, and J. Vahrenhold, "Detecting and understanding students' misconceptions related to algorithms and data structures," in *Proceedings of the 43rd ACM technical symposium on Computer Science Education*. ACM, 2012, pp. 21–26.

[26] K. Karpierz and S. A. Wolfman, "Misconceptions and concept inventory questions for binary search trees and hash tables," in *Proceedings of the 45th ACM technical symposium on Computer science education*. ACM, 2014, pp. 109–114.

[27] G. L. Herman, M. C. Loui, and C. Zilles, "Creating the digital logic concept inventory," in *Proceedings of the 41st ACM technical symposium on Computer science education*. ACM, 2010, pp. 102–106.

[28] K. C. Webb and C. Taylor, "Developing a pre-and post-course concept inventory to gauge operating systems learning," in *Proceedings of the 45th ACM technical symposium on Computer science education*. ACM, 2014, pp. 103–108.

[29] A. Luxton-Reilly, B. A. Becker, Y. Cao, R. McDermott, C. Mirolo, A. Mühling, A. Petersen, K. Sanders, J. Whalley *et al.*, "Developing assessments to determine mastery of programming fundamentals," in *Proceedings of the 2017 ITiCSE Conference on Working Group Reports*. ACM, 2018, pp. 47–69.

[30] A. E. Tew, "Assessing fundamental introductory computing concept knowledge in a language independent manner," Ph.D. dissertation, Georgia Institute of Technology, 2010.

[31] W. K. Adams and C. E. Wieman, "Development and validation of instruments to measure learning of expert-like thinking," *International Journal of Science Education*, vol. 33, no. 9, pp. 1289–1312, 2011.

[32] M. Shuttleworth, "Pretest-posttest designs," https://explorable.com/pretest-posttest-designs, retrieved Jun 18, 2019.

[33] D. Fisher and N. Frey, *Checking for Understanding: Formative Assessment Techniques for Your Classroom*. ASCD, 2014.

[34] A. Ahadi, R. Lister, and D. Teague, "Falling behind early and staying behind when learning to program," in *Proceedings of the 25th Psychology of Programming Conference, PPIG*, vol. 14, 2014.

[35] R. Bornat, S. Dehnadi, and D. Barton, "Observing mental models in novice programmers," in *24th Annual Workshop of the Psychology of Programming Interest Group, London., year=2012*.

[36] M. Ford and S. Venema, "Assessing the success of an introductory programming course," *Journal of Information Technology Education: Research*, vol. 9, pp. 133–145, 2010.

[37] M. E. Caspersen, K. D. Larsen, and J. Bennedsen, "Mental models and programming aptitude," in *ACM SIGCSE Bulletin*, vol. 39, no. 3. ACM, 2007, pp. 206–210.

[38] J. Henry and B. Dumas, "Towards the identification of profiles based on the understanding of programming concepts: the case of the variable," in *2019 IEEE Frontiers in Education Conference (FIE)*. IEEE, 2019, pp. 1–8.

[39] ——, "Developing an assessment to profile students based on their understanding of the variable programming concept," in *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, 2020, pp. 33–39.