Abstract: Control-flow statements often cause misunderstandings among novice computer science students. To better address these problems, teachers need to know the misconceptions that are typical at this stage. In this paper, we present the results of studying students’ misconceptions about control-flow statements. We compiled 181 questions, each containing an algorithm written in pseudocode and the execution trace of that algorithm. Some of the traces were correct; others contained highlighted errors. The students were asked to explain in their own words why the selected line of the trace was correct or erroneous. We collected and processed 10,799 answers from 67 CS1 students. Among the 24 misconceptions we found, 6 coincided with misconceptions from other studies, and 7 were narrower cases of known misconceptions. We did not find previous research regarding 11 of the misconceptions we identified.

Keywords: misconceptions; control-flow statements; introductory programming

1. Introduction

Nowadays, the growing need for software engineering (SE) specialists makes SE educational institutions improve both quantitatively and qualitatively. Mastering programming fundamentals remains one of the key requirements for successful software engineering education. However, this field introduces a lot of new concepts which are difficult to master [1,2]. This attracts a lot of attention from researchers, aiming at studying and improving introductory programming courses.

Most SE teachers witnessed students becoming confused while trying to understand the behavior of a computer program [3] or learn new concepts. Unfortunately, program execution in most of the IDEs is not visible until the user takes steps to visualize it (such as using the debugger), which many worse-performing students do not do, relying on their guesses instead. There is often little time to identify and eliminate the misconceptions of each student personally. So, correcting misconceptions should be a part of the regular educational process, as studying common misconceptions is useful. In software engineering, there have been a number of studies concerning misconceptions [4–7]; different methods have been developed for detecting misconceptions [8]. Some studies were conducted to fix misconceptions during classroom work [9]. Usually, methods for detecting and fixing misconceptions are based on domain knowledge and typical errors in students’ mental models.

A review of the related literature showed that little attention has been paid to the systematic study of misconceptions related to control-flow statements, unlike other topics such as the syntax of programming languages [10,11], memory allocation for objects and scalar values [12], passing parameters by value or by reference, and other advanced topics. The misconceptions concerning sequences, loops, and selection statements are mixed with misconceptions about other programming constructs in many studies. This does not allow gathering data aimed specifically at identifying misconceptions of control-flow statements and often makes misconception definitions too general.

The misconceptions described in the literature were formulated differently: some are very specific, and others are vague. The systematic formulation of a set of possible
Misconceptions is an important challenge helping to cover the subject. Misconceptions are identified in order to correct students’ mental models more easily, so a good misconception formulation should allow for identifying ways to correct them.

So, we see some misconceptions as too broadly formulated to be useful: their formulations may be too abstract or combine several different errors. Such high-level misconceptions are suitable only for superficial diagnostics and cannot be used for planning corrective pedagogical interventions. For example, the misconception “T2: Students misunderstand the process of while loop operation” [12] is too abstract and useful only for giving the student more tasks on while loops; it says nothing about how to correct their understanding of loops in general (e.g., it may relate to repeating the same lines of code, the role of loop conditions, etc.) and of the “while” loop in particular (e.g., the precondition).

Consider an example of composite misconception “M33: Loops terminate as soon as the condition changes to false” mentioned by [5,13]. Although it expresses a well-formed state of the student’s mind, it touches on several low-level misconceptions, such as breaking the sequentiality of actions within the loop body and ending the loop without explicit evaluation of the loop’s condition.

Obviously, it is impossible to consider and describe all the high-level misconceptions that students have (and may have in the future) about control-flow structures, which reduces the effectiveness of teaching aimed at detecting and removing high-level misconceptions. However, it is possible to reduce high-level errors to low-level misconceptions that relate to elementary rules for executing program code. In this case, the misconceptions should be aimed at explaining one trace line—i.e., one event during code execution. If we continuously correct simple misconceptions and explain to the students why this statement should be executed next, then, over time, we can deal with all higher-level misconceptions. For example, if the control condition of a selection statement is false, the “then” branch should not be executed; an attempt to execute the “then” branch can be attributed to a misconception related to the consequences of the selection statement’s condition evaluating to false. An example of a high-level misconception about this is “The ‘then’ branch is always executed” [13].

Our research questions are:

- RQ1. What undergraduate students’ misconceptions of control-flow statements can be identified by interpreting a single line of trace?
- RQ2. How do these misconceptions relate to the misconceptions known in the literature?

Misconceptions concerning sequences, loops, or selection statements are found in many studies intermixed with wrong ideas about similar topics; they are rarely studied on their own. That affects good coverage of the topic, e.g., Sirkiä and Sorva mention “Wrong False” but no kind of “Wrong True” [14]; Albrecht and Grabowski report “order of conditions” but ignore the possibility of skipping conditions [15].

Most of the researchers report only frequent misconceptions, which makes sense for teachers who have limited classroom time. However, the developers of software tutors should take into account less frequent misunderstandings as well to provide students with useful feedback in the maximum range of possible situations. Automatic feedback requires the misconceptions to be specific and precise. Some authors provide too broad formulations of misconceptions such as “Students misunderstand the process of while loop operation” [12], or “Difficulties in understanding the sequentiality of statements” [13], which gives no idea of what is actually wrong and what kind of help the student needs besides re-reading the relevant textbook chapter.

Too few misconceptions about loops have been found; they are mostly concerned with just “loop” or “while loop” [12]. We found no mention of other kinds of loops, such as “for loop” and “do-while loop”, and a variant of the latter, “repeat-until loop”, which has its condition inverted. The now popular family of “foreach” loops is closely related to the concept of collection and can easily be modeled by simpler loops, so we believe that “foreach” can be studied as a variant of the “for” loop.
In this study, we aimed to close this gap and research specifically the misconceptions about the control flow during program execution. We did it by showing an algorithm and its trace to students and asking them to explain why a particular trace line should be there (for correct traces) or should not be there (for incorrect traces). That limited the number of errors the students could make but let us get an insight into their thinking. As a result, we confirmed some already reported misconceptions, created detailed versions of some broadly generalized misconceptions, and identified misconceptions that we did not find in the relevant literature.

The identified misconceptions can help teachers plan their lessons and develop assessments; they also can be used to foster the development of software tutors providing automatic feedback: for example, to explain deviations from the solution path [16], to create sequences of follow-up questions [17], or to indicate invalid solution states [18].

This article is organized as follows. Section 2 describes related works from two points of view—methods for identifying misconceptions and examples of misconceptions related to selection statements and loops. Section 3 explains how the research on misconceptions was conducted. Section 4 describes the results of the study. In Section 5, we compare our findings with findings of other researchers and briefly discuss the limitations of our approach. Section 6 describes the threats to study validity. The Section 7 emphasizes the most important findings and shows the directions for further research. The list of the misconceptions found is provided in Appendix A.

2. Related Work

Identifying misconceptions is a developed field of study in many areas of education, including software engineering. Misconceptions of many different areas of software engineering education have been studied over time [4,19–21].

Some of those works are devoted to studying the influence of programming languages and teaching methods on understanding the basics of programming [22]. Others analyzed methods for detecting and reducing misconceptions [8,9]. There are two main aspects of the reviewing literature about misconceptions in software engineering education: the methods used and misconceptions in related areas that can be compared to our findings.

2.1. Methods of Studying Misconceptions

Identifying misconceptions requires collecting representative data on students’ thinking. Two basic approaches are to study the students’ thinking (mainly using structured and unstructured interviews) or the artifacts that students produce (program code, result prediction, code-tracing problems).

2.1.1. Interview

Various forms of interviews were used to study misconceptions such as verbal communication about assignments [23], “explain in plain English” [3], etc. Researchers have used the interview as a method of detecting students’ misconceptions [24,25]. In particular, Kaczmarczyk et al. [12] conducted interviews with students, encouraging a student to think aloud. The communication protocols were recorded and handed over to the specialists who analyzed the interview results independently to avoid bias. The assessment results for each answer were discussed, and the researchers came to a general conclusion on the type of misconception the student had.

This method allows in-depth insight into the students’ thinking. Still, it requires significant work to process each interview, so the number of participants is usually small, which threatens the validity of the results. One of the ways of increasing the number of participants is by structuring the interviews into a series of free-text responses to predefined questions. Further, it is possible to survey teachers (instead of students) about the misconceptions they encounter [26,27].
2.1.2. Analysis of Code Written by Students

This method relies on the results of code-writing assessments [28–32]. In their study, Ref. [15] used a bank of program code submitted by students to an automated learning environment. The code submissions were tested automatically, and the ones that did not produce the required answer (or the code did not compile) were chosen for further analysis. Each researcher assessed them.

This method allows analyzing bigger banks of code but does not give insight into what the student thought while writing the analyzed code.

2.1.3. Result-Prediction Problems

A result-prediction problem requires the student to imagine the execution of a program (or parametric function) and predict its output [13] or return value [6,7].

The method of mining misconceptions from students’ predictions is based on guessing a mental model that, with the given program, would produce the prediction obtained. That is not always possible, so only a small fraction of answers is normally used. The advantage of this method, as well as of the previous method, is a significantly higher number of participating students.

2.1.4. Code-Tracing Problems

To understand how, in the student’s opinion, a program (or algorithm) works, the researchers can ask the student to build an execution trace of the given program. Sirkiä and Sorva asked students to simulate the execution of a small program by manually performing all actions of the program in the graphical interface of a visual simulator [5,14]. It concerned the data about values, operators, function calls, parameters passed to functions, call stack, changes in the execution position, etc. The researchers analyzed obtained traces of program code execution by each student. They reported, among other things, patterns of erroneous behavior, probably related to the complexity of the simulator’s user interface (UI). To reduce the cognitive load on the student, this approach can be enhanced by using simulators, focusing only on one aspect of the program behavior [33].

2.1.5. Combined Methods

Some studies combine different methods: for example, Velázquez-Iturbide [29] analyzed the code created by the students in conjunction with free-text comments written by them in their code.

2.2. Misconceptions Related to Flow of Control

Kaczmarszyk et al. [12] identified several misconceptions; some of them concern iterations and loops. The closest to our topic is: “Students misunderstand the process of while loop operation”. However, that misconception is formulated very broadly and so is of little use.

Albrecht and Grabowski [15] identified 50 misconceptions related to programming. The following misconceptions are related to control-flow statements:

- “spurious/missing text/code fragment”;
- “missing condition” (it is unclear whether it applies only to loops or to “if-else” as well);
- “missing loop”;
- “order of conditions”;
- “code outside of function block”;
- “infinite loop”.

Swidan et al. [13] studied 11 misconceptions from the big list of misconceptions compiled by Sorva [5], which contained more than 160 items. These misconceptions were better formulated and more detailed. Some of them concern control-flow statements, for example:
• 23-Ctrl “Difficulties in understanding the sequentiality of statements”;
• 24-Ctrl “Code after ‘if’ statement is not executed if the ‘then’ clause is.”;
• 26-Ctrl “A false condition ends program if no ‘else’ branch exists”;
• 27-Ctrl “Both ‘then’ and ‘else’ branches are executed”;
• 28-Ctrl “The ‘then’ branch is always executed”;
• 30-Ctrl “Adjacent code executes within loop”;
• 31-Ctrl “Control goes back to start when a condition is false”;
• 33-Ctrl “Loops terminate as soon as condition changes to false” (interrupting iteration/skipping the rest of iteration).

Sekiya and Yamaguchi [6] conducted a study where students were asked to predict the results of performing small functions involving loops. The closest of the seven identified misconceptions to our research is “NFL: Neglect(ignore) For Loop” (the control condition was not evaluated at the beginning of the loop or after the iteration).

Sirkiä and Sorva [14] identified 11 misconceptions by analyzing student-generated execution traces. Some of them are related to the topics of our research, e.g., “Wrong branch”, “Wrong False”.

3. Method

Our goal was to collect students’ thoughts on how control-flow statements affect program execution. To do this, we asked them in a free-text form why the particular lines of code should or should not be in the specified positions, then marked and analyzed their answers. The scheme of our study is shown in Figure 1.

Figure 1. Method flowchart: UML activity diagram of studying misconceptions from students’ responses.

3.1. Data Collection

To collect students’ thoughts, we conducted a test using questions with free-text answers. Each question showed a small algorithm and its execution trace. One of the lines
in the trace was highlighted: the students had to explain why the highlighted line was correct or incorrect. We created and verified 181 questions on the reasoning for the traces of code execution (see Table 1). The questions can be divided into four categories as follows:

- Selections and sequences (correct traces) (e.g., the question shown in Figure 2);
- Selections and sequences (incorrect traces);
- Loops (correct traces);
- Loops (incorrect traces), see the example in Figure 3.

For each question category, several small algorithms were compiled—various forms of selection statements (one to three “if”, “if else” branches, with or without the “else” branch), and loops (“while”, “do-while”, “for”, “foreach”). Some questions about loops contained selection statements within the loop. Some of the algorithms have been used to create both correct and incorrect traces.

For each algorithm, a series of correct execution traces were created automatically by setting values for control conditions of selection statements and loops (we tried to keep traces short but non-trivial). A trace was represented as a sequence of acts of execution. Execution of simple statements was represented as a single act; the beginnings and ends of complex statements (selection statements and loops) and their parts (branches and iterations) comprised separate acts of execution to show nesting in the trace. That allowed us to verify how students understood statement nesting, which was rarely explicitly verified in previous studies. Statement nesting was shown by indentation and curly braces (for example, in Figure 3, the statement “wait” is inside the loop, but the statement “wait_seconds(3)” is not). The acts were supplemented with information about the values of control conditions and how many times they were executed (see Figure 3). Incorrect traces were created manually by modifying correct traces: deleting, duplicating, or moving acts of the original trace. Some correct traces were used in several questions; different lines of that trace were highlighted.

The questions used a pseudo-code notation, which was close to the C programming language the students studied. We did not specify expressions, replacing them with labels to avoid distracting students from the study topic—control-flow statements—and also avoid muddying the results with expression evaluation problems. So we replaced them with words and simple phrases resembling actions and conditions. The only exceptions were control conditions of selection statements and loops, which affected the control flow. The students saw the results of the control conditions’ evaluation in the relevant trace lines (for example, see the third trace line in Figure 2). The students were provided with samples explaining the pseudo-code notation of the algorithms and trace lines before the beginning of the study.

Table 1. Test setting and answers collected.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Selection</th>
<th>Loops</th>
<th>Total:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithms</td>
<td>6</td>
<td>9</td>
<td>15</td>
</tr>
<tr>
<td>Trace kind</td>
<td>correct</td>
<td>erroneous</td>
<td>correct</td>
</tr>
<tr>
<td>Traces</td>
<td>7</td>
<td>7</td>
<td>14</td>
</tr>
<tr>
<td>Questions</td>
<td>23</td>
<td>54</td>
<td>76</td>
</tr>
<tr>
<td>Respondents</td>
<td>67</td>
<td>67</td>
<td>63</td>
</tr>
<tr>
<td>Answers given</td>
<td>1539</td>
<td>3602</td>
<td>4254</td>
</tr>
<tr>
<td>Answers with misconceptions</td>
<td>32</td>
<td>85</td>
<td>21</td>
</tr>
</tbody>
</table>
The questions were used in a summative assignment in the “Programming basics” course for the first-year undergraduate students of Volgograd State Technical University majoring in software engineering. The test was conducted in a remote mode because of the COVID-19 lockdown. We analyzed the anonymized results of that assignment.

A total of 10,799 answers were received from 67 students.

3.2. Data Analysis

The analysis was based on a hybrid inductive–deductive approach [34]. All the researchers had experience in teaching introductory programming courses as professors and teaching assistants teaching the “Programming basics” course for years.

First, we analyzed a sample of responses to 23 questions from different categories (921 responses in total), identifying and clustering frequent errors. Based on that clustering, a preliminary list of misconceptions that were possible to identify with the created questions was compiled. The findings were formulated in terms of low-level misconceptions about the trace acts ordering, i.e., their mutual position, the absence of a necessary act, or the presence of an extra act. For example, suppose a student places the initialization of a FOR loop after an iteration, i.e., later than its usual location at the beginning of the loop. In that case, we consider this move as both LOOP8 (missing, skipping initialization) and LOOP6 (unexpectedly late initialization) misconceptions (see Table A1). Both of these elementary problems can arise separately because...
of the step-by-step nature of the trace construction and should be explained separately. Researcher 1 verified the resulting list and made corrections. The resulting list contained 16 misconceptions about selection, 12 about loops, and 9 other misconceptions.

After that, the collected answers were analyzed. Out of 2370 incorrect answers, we selected for further analysis 258 potentially interesting answers (11%) that could expose misconceptions. The remaining 89% of incorrect answers were excluded as irrelevant because they were incomplete, contradictory, lacked clarity, or were not related to answering the relevant questions (there were even 5 jokes).

Then, the remaining 258 answers were analyzed and assigned the labels from the selected sample with the misconception type from the previously created list. For each answer, the status was determined: the answer demonstrates a misconception, the answer is correct (i.e., answers the question at least partially), or the answer is unclearly formulated. After a discussion, the team excluded about 30% of the responses as not having misconceptions (either correct or irrelevant answers).

The remaining 180 responses with misconceptions (1.7% of all responses collected) were agreed to contain a misconception by both of the researchers. They were independently labeled with the previously agreed misconception labels. The two researchers’ agreement about the kind of misconception calculates to be 92% (agreed on 166 responses, disagreed on 14 out of 180). After that, we compared the two lists and produced a final list, flagging the answers on which there was and was not agreement. The answers with no independent agreement were discussed with the research team and with an external expert (also a teacher assistant in an introductory course) to achieve a shared understanding of their interpretation. The Cohen’s Kappa metric [35] for the final agreement was 0.918, which is interpreted as almost perfect agreement.

Finally, for each misconception, we counted the number of students who expressed it at least once as a more fair measure of its frequency (the number of questions, answers to which can expose a given misconception, varied). We excluded the misconceptions that were expressed only by one student if they were not mentioned in the previous literature.

4. Results

Table A2 shows examples of students’ answers exposing misconceptions. The final list of misconceptions aligned with the findings from related work is shown in Table A1. It includes 22 misconceptions and two types of terminological substitution:

- 3 misconceptions about sequences (SEQ*);
- 10 misconceptions about selection statements (SEL*);
- 8 misconceptions about loops (LOOP*);
- 1 misconception about concept recognition (RECsl);
- 2 misconceptions about notation (NOT*) or used terms.

Six of the misconceptions we observed align with related work precisely. These are as follows:

- SEL1 matches “28-Ctrl” reported by [5] and is close to “Misc2: Wrong branch” by [14];
- SEL2 is effectively the same as “UI2: Failing to jump upon selection” [14];
- SEL3 matches “e26: order of conditions” [15];
- SEL6 is a generalized form of “27-Ctrl: Both then and else branches are executed” reported by [5];
- LOOP3 is a sub-type of “30-Ctrl: Adjacent code executes within loop” reported by [5];
- LOOP4 is close to “Extra Count” (an extra iteration) by [7].

Other misconceptions we found are specific cases of known misconceptions. For example, “T4: Students cannot trace code linearly” [12] and “23-Ctrl: Difficulties in understanding the sequentiality of statements” reported by [5] are covered in our misconceptions SEQ1, SEQ2, and SEQ3. Another general misconception “T2: Students misunderstand the process of while loop operation” [12] was clarified by LOOP1–LOOP4.
The misconception when the student does not attempt to repeat the control condition and loop body, reported by [6] as “NFL: Neglect(ignore) For Loop”, is close to LOOP2 and LOOP5. LOOP1 and LOOP2 are subclasses of “e45: missing condition” [15].

Some known misconceptions occurred rarely (i.e., were demonstrated by only one student):

- RECsl and SEL9 are kinds of “31-Ctrl: Control goes back to start when condition is false” [5,13];
- LOOP5 is a more general misconception than “Missing 0” (skipping the first iteration) by [7].

To the best of our knowledge, nine of the found misconceptions related to understanding how control-flow statements work have not been reported by other researchers. These include the following:

- SEL4, SEL10 are specific to “think aloud”/“write what you think” methods and unlikely to be found by other methods;
- LOOP6, LOOP7, LOOP8 are detectable by the “simulate execution” method too, but they are about “for” and “foreach” loops which received less attention from other researchers than “while” and “do–while” loops;
- SEL7 and SEL8 are only detectable if the researcher asks the students about the beginning and ending of each control-flow statement during execution, not just about executed lines, which was done rarely in the previous literature;
- SEL5, SEL9 can be detected by using the “simulate execution” method, but they were not reported in the literature (to the best of our knowledge); one possible explanation is the low number of tasks using selection statements with multiple branches that are required for detecting SEL5 and increase the possibility of detecting SEL9.

The two common misconceptions found (NOT1sl, NOT2pl) are related to terminology: students used the word “loop” while speaking about selection statements or the entire program. They can only be found using structured or unstructured free-text interviews; other methods do not expose the terms the students use, which may explain why they were not reported in the previous literature.

The misconception “Missed non-loop” reported in [7] was not found in our study.

5. Discussion

In this study, we researched misconceptions about control-flow statements (selection statements: “if-else”, loops: “while”, “do–while”, “for”, “foreach”) by analyzing almost 10,800 short open-ended answers from students in the “Programming basics” course. The questions showed students an algorithm and its trace and asked them to explain why the particular trace line is correct or incorrect.

The vast majority of the answers we collected were at least partially correct. That indicates a generally good understanding of control-flow statements by first-year undergraduates majoring in software engineering. Many of the erroneous answers were irrelevant or incomprehensible, which indicates that teachers need to pay more attention to teaching students to formulate their thoughts clearly. Given our large initial base of answers, the number of answers exposing misconceptions remained considerable.

- RQ1. What undergraduate students’ misconceptions of control-flow statements can be identified by interpreting a single line of trace?

During this study, we formulated a systematic list of 24 control-flow misconceptions regarding a single line of code, including 3 misconceptions about sequences, 10 about selection statements, 8 about loops, 1 about concept recognition, and two cases of misuse of domain terms. The most frequently observed misconceptions were as follows:

1. Not considering the truth of a control condition as the cause for the execution of its branch—demonstrated by 10 students out of 67 participants;
2. Conditions of a selection statement are checked in the wrong order—demonstrated by 9 students;
3. A “for” loop is initialized after its iteration—demonstrated by 8 students;
4. Calling the selection statement a loop—demonstrated by 8 students;
5. Failing to take into account the end of a block—demonstrated by 7 students;
6. Starting a ‘then’ branch after a false condition—demonstrated by 7 students;
7. Skipping the update step of a “for” or “foreach” loop—demonstrated by 7 students;
8. Two or more branches of one selection statement are executed—demonstrated by 6 students;
9. Action, which is placed after the selection, is considered belonging to its branch—
demonstrated by 6 students;
10. Continuing a loop after its control condition evaluates to false—demonstrated by 6 students;
11. Action, which is placed after a loop, is considered belonging to the loop—demonstrated by 6 students.

Out of the most frequently observed misconceptions, five concern behavior of selection
statements (primarily multi-branch selection), four concern loops (primarily “for” and
“foreach” loops), one concerns sequences, and one concerns terminology usage. This
indicates that the areas which most need additional attention during novice programming
courses are selection statements with more than two branches and “for” loops which are
more complex than “while” and “do-while” loops.

- RQ2. How do these misconceptions relate to the misconceptions known in the literature?

Out of 24 misconceptions that we observed, 13 are similar to the misconceptions from
previous research (6 coincide with previously found misconceptions, while 7 are narrower
cases of known misconceptions), and 11 were not found by us in the previous literature.
The misconception “Missed non-loop” (missing a trace line) that we found reviewing other
studies was relevant to our method, but no student demonstrated it during our study.

The newly found misconceptions are as follows:
1. Not considering the truth of a control condition as the cause for the execution of
   its branch;
2. Checking a condition after executing a branch;
3. Action, which is placed after the selection, is considered belonging to its branch;
4. Exiting a selection without executing a branch or checking all the conditions;
5. Multiple checks of the same condition in one selection statement;
6. Executing a wrong branch after checking a condition;
7. A “for” loop is initialized after its iteration;
8. Skipping the update step of a “for” or “foreach” loop;
9. Skipping the initialization of a “for” loop;
10. Calling the selection statement a loop;
11. Calling the program a loop.

6. Threats to Validity

The main threat to the validity of this research is that it was conducted in one
educational institution, and all the participants were undergraduate students majoring in
computer science or software engineering. It is possible that K-12 students and non-CS
majors will have different distributions of the frequency of these misconceptions. Because
of the method we used (free-text answers entered into an automatic test program), some
of the information could be lost if the students did not consider it valuable enough to
report. The test was conducted in a remote mode, which is different from the classroom
situation, so the students could not ask their teachers for help during it. However, this
might have allowed them to expose their misconceptions more freely. Another threat to
validity is interpreting students’ free-text answers. Some of them were worded so vaguely
that it was impossible to say what the student meant. We excluded too vague answers from
misconception identification, which might have affected the observed frequency of the
misconceptions. The test was analyzed anonymously so that the students would not fear
the results affecting their course grades; that did not allow us to interview the participants about their vague answers.

7. Conclusions

This study provides a new perspective on misconceptions in computer science education. First of all, in the previous research, the methods allowing insight into students’ thinking (interview, focus groups) did not allow analyzing a large sample of answers, which can lead to missing some of the less common misconceptions. The methods allowing a large number of participants (code analysis, problem solving) allow the researchers to work only with the artifacts of student’s work, which limits the number of misconceptions that can be detected. By labeling and analyzing free-text responses to predefined questions, we were able to work with a larger student sample while letting them answer in their own words.

While undergraduate CS majors are common subjects in studies of misconceptions of novice programmers, most of the existing studies span all the misconceptions encountered in CS0/CS1 course. None of them to our knowledge concentrated on studying misconceptions of control-flow statements in particular. That often led to broad formulations of found misconceptions and could cause some of them to be omitted. Our research closes this gap. We created a set of questions requiring students to explain lines of program traces, which were specifically aimed at verifying understanding of control-flow statements. This allowed us to find 11 new misconception and narrow definitions of 7 already-known misconceptions, along with confirming 6 previously known ones.

Among the found misconceptions, the two most frequently observed misconceptions (and three out of the seven most frequently observed misconceptions) concerned multi-branch selection statements. This shows that learning primarily one-branch and two-branch “if-else” statements is not enough and more attention should be given to the behavior of more complex selection structures, including “else if” (or “elif”) clauses. Another two out of the seven most frequently observed misconceptions concern the behavior of “for” and “foreach” loops, which is more complex than the behavior “while” and “do-while” loops. “For” loops also need additional attention. A somewhat surprising finding is that students often called any non-linear algorithmic structure a loop, so terminology questions, stressing the difference between iterative and non-iterative algorithmic structures, are needed.

These misconceptions can be used in developing web-based intelligent tutoring systems for teaching programming such as Problets [36], HowItWorks or CompPrehension [17]. Those systems rely on the automatic determining and classifying of students’ errors and providing relevant error messages to correct students’ misconceptions. What they require from misconception studies is different. The frequency of misconceptions does not matter for automatic tutors because they do not have the time restrictions of human teachers; automatic tutors can address even rare misconceptions to help more students. However, developing intelligent tutoring systems requires fine-grained lists of misconceptions regarding a particular topic, which is in line with the results of this study.

For further research, we plan to analyze the collected answers according to the information they contain: we plan to identify atomic information pieces that should be present in the answer to each question and determine the pieces that were often omitted. We also plan to use these misconceptions to enhance automatic tutors for introductory programming courses with advanced feedback for students. Common misconceptions can also be used to fine-tune adaptive exercises in automatic tutors to make sure each of them was verified during the exercise.

Author Contributions: Conceptualization, O.S.; methodology, O.S.; validation, M.D.; formal analysis, O.S. and M.D.; resources, O.S.; data curation, M.D.; writing—original draft preparation, O.S.; writing—review and editing, O.S.; visualization, M.D.; funding acquisition, O.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research has received no funding.
Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Ethical review and approval is not required, because this study involved the analysis of data being based on voluntary participation and having been properly anonymized. The research presents no risk of harm to subjects.

Data Availability Statement: The data is available from the corresponding author upon reasonable request. The data consists of free-text answers, which are not in English language.

Acknowledgments: We thank Fomichev Eugeny for helping us analyze conflicting labeling.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations
The following abbreviations are used in this manuscript:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td>Computer science</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Developer Environment</td>
</tr>
<tr>
<td>SE</td>
<td>Software engineering</td>
</tr>
</tbody>
</table>

Appendix A. Misconceptions and Responses Labeling

Table A1. List of detected misconceptions.

<table>
<thead>
<tr>
<th>Label</th>
<th>Misconception</th>
<th>Count of Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQ1</td>
<td>Failing to take into account the end of a block: The student thinks that a block or function has not been completed, although it has ended.</td>
<td>7</td>
</tr>
<tr>
<td>SEQ2</td>
<td>Action, which is placed after a sequence, is executed inside the sequence: Inside the block, the student executes an action that is placed after the block.</td>
<td>3</td>
</tr>
<tr>
<td>SEQ3</td>
<td>Actions are executed in the wrong order: The student executes actions inside a block in a different order than they are placed in the sequence.</td>
<td>1</td>
</tr>
<tr>
<td>SEL1</td>
<td>A “then” branch is always executed: The student thinks that a selection-statement branch can be executed without evaluating its condition first.</td>
<td>3</td>
</tr>
<tr>
<td>SEL2</td>
<td>Starting a “then” branch after a false condition: The student tries to start a selection-statement branch even though its condition is false.</td>
<td>7</td>
</tr>
<tr>
<td>SEL3</td>
<td>Conditions of a selection statement are checked in the wrong order: The student thinks that, in a multi-branch selection statement, conditions can be checked not in the order they are placed in the statement</td>
<td>9</td>
</tr>
<tr>
<td>SEL4</td>
<td>Not considering the truth of a control condition as the cause for the execution of its branch: When asked why a branch of the selection statement has started, the student does not mention the value of its control condition as the reason for its execution.</td>
<td>10</td>
</tr>
<tr>
<td>SEL5</td>
<td>Checking a condition after executing a branch: The student thinks that a condition can be evaluated after the corresponding branch of the selection statement is completed.</td>
<td>5</td>
</tr>
<tr>
<td>SEL6</td>
<td>Two or more branches of one selection statement are executed: The student thinks that more than one branch can be executed during one execution of a selection statement.</td>
<td>6</td>
</tr>
<tr>
<td>Label</td>
<td>Misconception</td>
<td>Count of Students</td>
</tr>
<tr>
<td>-------</td>
<td>---------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>SEL7</td>
<td>Action, which is placed after the selection, is considered belonging to its branch: Inside a selection-statement branch, the student executes an action that is placed after the selection statement.</td>
<td>6</td>
</tr>
<tr>
<td>SEL8</td>
<td>Exiting a selection without executing a branch or checking all the conditions: The student thinks that a selection statement can finish before checking all its conditions and/or without executing a branch whose condition is true.</td>
<td>2</td>
</tr>
<tr>
<td>SEL9</td>
<td>Multiple checks of the same condition: The student thinks that in a selection statement, a branch condition can be evaluated several times.</td>
<td>4</td>
</tr>
<tr>
<td>SEL10</td>
<td>Executing a wrong branch after checking a condition: The student thinks that a different branch can be executed after a selection-statement condition evaluates to true.</td>
<td>3</td>
</tr>
<tr>
<td>LOOP1</td>
<td>Missing first condition test in a precondition loop: The student thinks that the first iteration of a precondition loop can begin without evaluating the loop condition first.</td>
<td>4</td>
</tr>
<tr>
<td>LOOP2</td>
<td>Continuing a loop without checking its condition: The student thinks that the next iteration of a loop can begin without evaluating the loop condition after the end of the previous iteration.</td>
<td>4</td>
</tr>
<tr>
<td>LOOP3</td>
<td>Continuing a loop after its control condition evaluates to false: The student thinks that a loop iteration can begin if the loop condition is false (for “while”, “do-while” and “for” loops).</td>
<td>6</td>
</tr>
<tr>
<td>LOOP4</td>
<td>Action, which is placed after a loop, is considered belonging to the loop: Inside the body of a loop, the student executes an action that is placed after the loop.</td>
<td>6</td>
</tr>
<tr>
<td>LOOP5</td>
<td>Skipping an iteration when the loop condition is true: The student thinks that the computer can proceed to the next action of the loop (e.g., checking the condition again) without executing the loop’s iteration.</td>
<td>5</td>
</tr>
<tr>
<td>LOOP6</td>
<td>A “for” loop is initialized after its iteration: The student thinks that the initialization step of a “for” loop can be executed after the iteration finishes.</td>
<td>8</td>
</tr>
<tr>
<td>LOOP7</td>
<td>Skipping the update step of a “for” or “foreach” loop: The student thinks that the update step of a “for” or “foreach” can be omitted after an iteration finishes.</td>
<td>7</td>
</tr>
<tr>
<td>LOOP8</td>
<td>Skipping the initialization of a “for” loop: The student thinks that the initialization step of a “for” loop should not be executed when the loop starts.</td>
<td>5</td>
</tr>
<tr>
<td>RECsl</td>
<td>Perceiving a selection statement as iterative: The student thinks that a selection statement can have several iterations during a single execution of the statement.</td>
<td>1</td>
</tr>
<tr>
<td>NOT1sl</td>
<td>Calling the selection statement a loop: When explaining the behavior of the selection statement, the student calls it a loop.</td>
<td>8</td>
</tr>
<tr>
<td>NOT2pl</td>
<td>Calling the program a loop: When explaining the behavior of the program, the student calls it a loop.</td>
<td>3</td>
</tr>
</tbody>
</table>
Table A2. Examples of labeling students answers.

<table>
<thead>
<tr>
<th>Misconception Assigned</th>
<th>Question Description</th>
<th>Answer Template</th>
<th>Correct Answer</th>
<th>Student’s Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEL1 (q. #139)</td>
<td>A loop having a selection with the target act iterates two times.</td>
<td>The act “well_done” occurs exactly twice in the trace, because...</td>
<td>both times when the selection statement was executed in the loop, its condition evaluated to true.</td>
<td>the action must be performed once in each iteration, so there are two iterations.</td>
</tr>
<tr>
<td>SEL4 (q. #8)</td>
<td>A branch’s condition has been evaluated to true, and the branch started</td>
<td>The branch should begin here because...</td>
<td>its condition evaluates to true.</td>
<td>it stands after checking the condition.</td>
</tr>
<tr>
<td>SEL5 (q. #31)</td>
<td>A branch’s condition has been evaluated to false and evaluated again.</td>
<td>The condition “now_green” is wrong in this place, because ...</td>
<td>the selection should check the conditions one by one.</td>
<td>the “else” branch should have been executed first, and only then the condition should be checked.</td>
</tr>
<tr>
<td>SEL9 (q. #30)</td>
<td>A branch’s condition has been evaluated to false and is evaluated again.</td>
<td>The condition “now_red” is wrong in this place, because...</td>
<td>the selection should check its condition only once.</td>
<td>it must appear before the previous condition.</td>
</tr>
<tr>
<td>LOOP4 (q. #168)</td>
<td>Action “go” stands after the end of a “while” loop; the loop tries to end without evaluating its condition.</td>
<td>The end of the loop is wrong in this place, because...</td>
<td>the condition of the loop must be evaluated first.</td>
<td>it should end after the “go” act.</td>
</tr>
<tr>
<td>LOOP6 (q. #178)</td>
<td>A “for” loop’s iteration begins without checking its condition.</td>
<td>The iteration should not begin here because ...</td>
<td>the condition of the loop has not been evaluated.</td>
<td>2nd initialization was not performed.</td>
</tr>
</tbody>
</table>

References

2. Shi, N. Improving Undergraduate Novice Programmer Comprehension through Case-Based Teaching with Roles of Variables to Provide Scaffolding. *Information* 2021, 12, 424. [CrossRef]


31. Sanders, K.; Thomas, L. Checklists for Grading Object-Oriented CS1 Programs: Concepts and Misconceptions. ACM SIGCSE Bull. 2007, 39, 166–170. [CrossRef]


**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.