systems

Article

A Classification of Adaptive Feedback in Educational Systems for Programming

Nguyen-Thinh Le

Humboldt-Universität zu Berlin, Department of Computer Science, Research Group "Computer Science Education/Computer Science and Society", Unter den Linden 6, 10099 Berlin, Germany; nguyen-thinh.le@hu-berlin.de; Tel.: +49-30-2093-3186

Academic Editor: Ockie Bosch
Received: 23 December 2015; Accepted: 5 May 2016; Published: 23 May 2016

Abstract: Over the last three decades, many educational systems for programming have been developed to support learning/teaching programming. In this paper, feedback types that are supported by existing educational systems for programming are classified. In order to be able to provide feedback, educational systems for programming deployed various approaches to analyzing students' programs. This paper identifies analysis approaches for programs and introduces a classification for adaptive feedback supported by educational systems for programming. The classification of feedback is the contribution of this paper.

Keywords: adaptive feedback; educational systems for programming; program analysis techniques

1. Introduction

Programming skills are becoming a core competence for almost every profession [1] and thus, Computer Science education is being integrated in the curriculum of almost every study subject. However, it is well-known that programming courses, which constitute an indispensable part of studies related to Computer Science, are considered a difficult subject by many students [2,3]. Addressing this problem, various approaches have been proposed to help students learn solving programming problems. One of the solutions to these problems is to deploy effective technology-enhanced learning and teaching approaches. As a consequence, researchers have identified this gap and have been developing various types of educational systems for programming that are able to analyze students' programs and provide feedback.

Feedback is one of the most powerful ways to enhance learning. In the behavioristic view, through feedback (stimulus), the behavior (output) of a student can be shaped. In an operant conditioning learning setting, feedback can serve as positive or negative reinforcement that shapes behaviors and thus, support learning [4]. In the cybernetic view, which assumes that a monitor compares the state of a system at various times with some standard and a controller adjusts the system's behavior, the aim of feedback is to reduce the gap between the student's current and desired states of learning [5]. In the constructivist view, which assumes that knowledge is constructed by each individual, feedback provides help so that each individual student can overcome a problem and progress learning with individual activity, e.g., [6]. Narciss [7] developed an interactive tutoring feedback (ITF) model based on the view that a student is an active constructor of knowledge and feedback aims at providing students with formative or tutoring information on their current state of learning which help them to regulate their learning process. According to Narciss [7], feedback may have different functions: acknowledge, confirm, reinforce correct response or high quality learning outcomes, and promote the acquisition of the knowledge and cognitive operations necessary for accomplishing learning tasks. In addition, on the motivational level, it can encourage students in maintaining their effort and persistence.

Since this paper discusses about adaptive feedback, it is required to review the notions of adaptive feedback. According to Dempsey and Sales [8], "adaptive feedback, unlike adapted and generic feedback, is dynamic. As they [learners] work through the instruction, different learners will receive different information from the computer. The determination of what feedback is provided, when, and to whom is made by the designer after careful consideration of all available information" ([8], p. 166). This notion of adaptive feedback has a broad sense. That is, instead of giving a feedback such as "wrong" for different incorrect answers and "correct" for different correct answers, adaptive feedback would not only verify the correctness of an answer, but also should provide different information for different answers. Also in this broad sense, feedback is considered adaptive when "different learners receive different information" [9]. For example, in the programming domain, individual information in a feedback could be the location of the erroneous program code, the explanation of the wrongly applied programming concept, an individual hint for improving the student's solution such as "the program is wrong because the code line X violates the concept Y". That means, in order to be able to provide adaptive feedback in the broad sense, an educational system needs to have the intelligent capability to analyze the error(s) in the student's solutions. The representative educational systems for programming that support adaptive feedback in this sense include FIT Java Tutor [10], ITAP [11]. Some other researchers proposed another notion of adaptive feedback. "Learners differ from each other in many ways including prior knowledge, meta-cognitive skills, motivational and affective state, or learning strategies and styles. Thus, these are many individual factors that may influence how feedback is processed by each learner. Consequently, a variety of individual factors can be used to support the design of personalized feedback strategies and implementation of AESs (Adaptive Educational Systems) which can deliver feedback messages that are tailored to the characteristics of an individual learner or a category of learners" ([12], p. 59). In this sense, adaptive feedback is generated by an adaptive educational system that requires a student model to represent the student's characteristics (e.g., knowledge, meta-cognitive skills, affective state, learning strategies and styles) and the student model serves to distinguish the level among different students [13]. This is a strict sense of adaptive feedback and, according to this notion, an adaptive educational system may provide adaptive feedback on the correctness or quality of a response (e.g., correct/incorrect; excellent/poor) or an elaborated feedback (in terms of the feedback classification proposed by Narciss [7]) depending on the student's model. The student model and its adaptation algorithm determine the type of feedback as well as the feedback strategy. This notion of adaptive feedback is also referred to as "personalized feedback" [12]. The representative educational systems for programming that support adaptive feedback in this sense include efforts that gradually provide appropriate feedback information adapting to the students' needs, e.g., [12], and systems that provides feedback based on students' individual knowledge level, e.g., [14] or gender, e.g., [15].

In this paper, both senses of adaptive feedback are adopted to consider educational systems for programming. That is, adaptive feedback is provided differently for individual students by analyzing the student's action (e.g., student's solution attempt) and/or adaptive feedback may be provided by an education system that is based ona student model. Existing educational systems for programming are reviewed and feedback types supported by these systems are classified. The classification of adaptive feedback used in educational systems for programming distinguishes from the generic feedback classifications (e.g., Fleming and Levie [16], Economides [17], Narciss [7]) in that it serves only researchers and developers in the domain of programming to focus on specific types of feedback.

The remainder of this paper is structured as follows. The next section presents the method of identifying specific feedback types supported by educational systems for programming. A classification of feedback types for educational systems for learning programming is classified in Section 3. The impact of different feedback types is discussed in Section 4. Findings in the review of feedback types are summarized and the developed classification of feedback types for programming is compared with related works in Section 5. In addition, research directions are proposed in this section.

2. Method

With the purpose of classifying specific feedback types that are supported by educational systems for programming, scientific reports of systems that have been published since 1999 at scientific workshops, conferences, and in journals related to the areas learning science and computer technology for education are searched on the Internet. In addition, technical reports, Master's theses, and PhD dissertations that have passed the internal review process and have been published by their universities were taken into consideration. In addition to searching publications on the Internet, authors of the collected publications were contacted to ask for their latest evaluation reports. Only publications since 1999 were collected, because Deek and McHugh [18] published in 1998 a detailed survey on educational systems for programming.

For the collected publications, there are two criteria: (1) publications from 1999 to 2009 have to show an evidence of practical use in courses or a scientific evaluation of the proposed system; and (2) recent publications from 2010 have to document either an implementation or a scientific study of the proposed approach. The first criterion for publications assures that only systems that have been demonstrated in a practical or lab setting are taken into account. The second criterion allows us to consider recently developed systems.

In addition to criteria for publications, several criteria with respect to functionality have been specified to shape the space of reviewed systems. First, the functionality of reviewed systems has to be beyond how compilers (e.g., providing (detailed) information about syntactical mistakes), and integrated programming environments (e.g., supporting syntax highlighting such as Eclipse) work. This criterion filters out two types of systems, compilers and integrated programming environments. In addition, the systems to be reviewed must have at least one interactive functionality. Interactive activity means that the students have the option to produce something (e.g., an answer or a program) rather to consume something. That is, the systems have to offer students more interactive experiences than simply providing online learning materials and test items.

With respect to the learning objectives, educational systems for programming that deal with program code, either by accepting student's solutions written as source code, or by using source code as templates (e.g., implementing a fill-in-the-gap strategy), were considered. That means, this criterion selects only systems that focus on the coding phase of programming and opts out those systems that support students pursue other learning objectives in the analysis, planning/design, and test phases of programming, for example, [19–22].

The last criterion is selecting only educational systems that support adaptive feedback in either the strict sense or the broad sense discussed in the previous section.

In summary, in addition to technical reports, Master and PhD dissertations, publications from the following conference/workshop proceedings, journals, or books were collected:

- Conference on Intelligent Tutoring Systems
- ACM Conference on Innovation and Technology in Computer Science Education
- IEEE International Conference on Computational Intelligence and Multimedia Applications
- IEEE International Conference on Advanced Learning Technologies
- European Conference on Technology Enhanced Learning
- Central European Conference on Information and Intelligent Systems
- Workshop on Adaptive and Intelligent Web-based Education Systems
- International Journal of Human-Computer Studies
- International Journal of Artificial Intelligence in Education
- ACM Journal on Educational Resources in Computing
- Journal Machine Learning
- Journal of Systemics, Cybernetics and Informatics
- inroads—The SIGCSE Bulletin
- Book: Advances in Learning Processes

3. A Classification of Adaptive Feedback in Educational Systems for Programming

Before a specific adaptive feedback type used in educational systems for programming is specified, classifications of generic feedback are briefly reviewed. Based on those classifications, the specific classification of adaptive feedback in educational systems for programming is developed. Fleming and Levie [16] classified five types of feedback based on the function dimension. Confirmation feedback indicates whether a solution is correct or incorrect. Corrective feedback provides information about a possible correct response. Explanatory feedback explains why a response is incorrect. Diagnostic feedback attempts to identify misconceptions by comparing the student's solution with common errors. Elaborative feedback provides additional related information. While Fleming and Levie [16] were based on the function dimension to classify feedback, Economides [17] was based on the content of feedback and distinguished seven types of (cognitive) feedback (actually, Economides [17] included a class of "no feedback" in the classification of feedback. We do not mention this class because feedback of this class does not contain any content): (1) Yes/No feedback; (2) answer until correct (feedback asks the student to try again until she answers correctly); (3) correct answer (feedback indicates the correct answer); (4) topic contingent (feedback elaborates on the general topic); (5) response contingent (feedback gives explanation); (6) bug related (feedback presents common errors made by students); and (7) attribute isolation (feedback highlights the central attributes of the target concept). Narciss [7] also classified feedback types based on the content dimension: (1) knowledge of performance (e.g., percentage of correctly solved tasks; number of errors; grade); (2) knowledge of result (information about the correctness (e.g., "correct"/"incorrect") or quality of the actual answer or outcome); (3) knowledge of the correct response (feedback content is a correct response or a sample solution to a given task); and (4) elaborated feedback (containing additional information besides knowledge of result or knowledge of the correct response (e.g., hints, guiding questions, explanations, worked examples)). The last feedback type (elaborated feedback) is divided into at least five sub-types, according to Narciss [7], depending on the content of elaborated feedback: (a) knowledge about task constraints; (b) knowledge about concepts; (c) knowledge about mistakes; (d) knowledge about how to process the task; and (e) knowledge about meta-cognition. Some of these subtypes of elaborated feedback proposed by Narciss [7] are in accordance with the feedback types 4–7 of the feedback classification proposed by Economides [17]. Across the three classifications, there are commonalities as Table 1 indicates.

Table 1. Commonalities between the three classifications of feedback.

Fleming and Levie [16]	**Economides [17]**	**Narciss [7]**
Confirmation	Yes/No	Knowledge of result
Corrective feedback	Correct answer	Knowledge of the correct response
Explanatory feedback	Response contingent	Elaborated feedback
Diagnostic feedback	Bug related	Elaborated feedback
Elaborative feedback	Topic contingent	Elaborated feedback

Are these types of feedback adaptive? If we adopt the broad sense of adaptive feedback, then corrective, explanatory and diagnostic feedback may be considered adaptive, because its information could be different for different students' solutions and confirmation feedback is not adaptive. If we adopt the strict sense of adaptive feedback, *i.e.*, an educational system is able to adapt different types of feedback according to individual student's preferences, knowledge level or learning style, then confirmation feedback could also be considered adaptive, because the educational system may have a student model that represents the preferences, the knowledge level or the learning style of a student.

In each of the following subsections, educational systems for programming that support a specific type of feedback are reviewed. The review of each system consists of two parts: an example of feedback (if an example of feedback supported by each system is available in the literature) and a brief description of the analysis technique that was developed to provide feedback. In Table 2, educational systems for programming that are mentioned in this paper are summarized and a representative reference for each system is cited (because each system may have been reported in several papers).

Table 2. Impact of the educational systems on students' programming.

	Evaluation		**Feedback Types**				
System	**Method**	**Result**	**Yes/No**	**Syntax**	**Semantic**	**Layout**	**Quality**
Ask-Elle [23]	Qualitative evaluation	+ "the feedback services are adequate"		Y	Y, I, C		
ELM-ART Weber and Brusilovsky ([14])	Quantitative and qualitative	+ "learners without previous programming knowledge profited more from the learning system in the ELM-ART group than . . . "			Y, C		
ELP [24]	Qualitative evaluation	+ "the system would help them [students] in thinking abstractly and develop their problem solving skills"		Y			Y
FIT Java Tutor [10]	**Technical evaluation**				Y, C		
Goshi's ist [25]	Qualitative evaluation	+ "More than 60% students thought these 3 functions [advice, detail, colour] help"			Y, C		
Hong's PROLOG [26]	**Technical evaluation**				Y, I, C		
INCOM [27]	Quantitative and qualitative	+ "the system did contribute to the improvement of the students' programming skills"			Y, I, C		
ITAP [11]	**Technical evaluation**				Y, C		
JACK [28]	Quantitative and qualitative	+ "Students and tutors got quickly used to use the traces at least for a short glance when reading the feedback messages"			Y, C		
JavaBugs [29]	**Technical evaluation**				Y, I, C		
JITS [30]	Qualitative evaluation	+ "Feedback mechanism – It provides hints quickly and to the point", "JITS helps students solve syntax and logic errors while developing a solution to a problem"			Y, C		

Table 2. *Cont.*

	Evaluation		Feedback Types				
System	**Method**	**Result**	**Yes/No**	**Syntax**	**Semantic**	**Layout**	**Quality**
J-Latte [31]	Quantitative and qualitative	+ “The participants’ knowledge did improve while interacting with the system, and the subjective data collected shows that students like the interaction style and value the feedback obtained”			Y, C		
jTutor [32]	**Technical evaluation**				Y, C		
Ludwig [33]	Quantitative and qualitative	+ “Seven students were able to finish the problem, one was almost completed, and two students were somewhat flummoxed and did not complete it”, “7 students thought that the style checker was an “excellent” idea; 2 thought it was a “good” idea; 1 thought it was an “okay” idea.”		Y		Y	
MEDD [34]	**Technical evaluation**				Y, I, C		
M-PLAT [35]	**Technical evaluation**		Y				
QuizJet [36]	Quantitative evaluation	+ “working with the system students were able to improve their scores on in-class weekly quizzes”	Y				
QuizPack [37]	Quantitative evaluation	+ “The students’ work with QuizPACK significantly improved their knowledge of semantics and positively affected higher-level knowledge and skills”	Y				
Radosevic’s system [38]	Quantitative evaluation	+ “the tools for program analyses and debugging help them to find the cause of their errors”			Y, C		
VC PROLOG [39]	**Technical evaluation**			Y	Y, I, C	Y	

3.1. Yes/No Feedback

The scarcest feedback type is confirmation feedback. Typical systems that provide this feedback type include QuizJet [36], QuizPack [37], and M-PLAT [35]. QuizJet and QuizPack provide individualized assessment questions in Java and C, respectively. “QuizJET [. . .] was designed as a generic system to author, deliver, and assess a range of parameterized questions for Java programming language. QuizJET can work in both assessment and self-assessment modes and is able to cover a whole spectrum of Java topics from Java language basics, to such critical topics as objects, classes, interfaces, inheritance, and exceptions.” [36]. QuizJET generates an individual Java program using the generic system and asks the student a question of analyzing the presented program to find the result for a specific variable, for instance, “*What is the final value of RESULT?*” (RESULT is a variable name in the given program). Then, QuizJET evaluates the student’s answer and presents a confirmation message “Wrong/Correct” and corrective feedback “Your answer is XXX. Correct answer is YYY” [36]. QuizPack works also in the same principle for the programming language C. Since both QuizJET and QuizPack are able to generate parameterized questions, the systems provide adaptive feedback.

M-PLAT does not provide multiple-choice test or fill-in-slot exercises, rather the system requests students to submit a whole program, then the system evaluates the student’s program and returns whether the student’s program is successful or not. Using M-PLAT to develop a program, “when a student has finished its program, the source code is sent to the server to be compiled and evaluated. Then, a response with the results is sent back to client-side application. After that, the student will receive one of the following messages: “Compilation errors have been found”, “There are no compilation errors, but the provided program is not correct”, “The program is correct” ([40], p. 128). Such feedback messages are of type confirmation feedback. Since M-PLAT is an adaptive educational system as the authors claimed “M-PLAT will adapt the level of requirements to the learning pace of each student” ([40], p. 127), the feedback messages are provided individually to students.

The common analysis technique deployed by the educational systems for programming reviewed above is that they compare the student’s solutions with pre-specified correct values. If the student’s solution does not match pre-specified correct values, then the system will indicate that the student’s solution is not successful. Otherwise, the system will present a feedback message indicating that the student’s solution is correct.

3.2. Syntax Feedback

Syntax feedback is based on the compiler’s output. Most systems just return the compiler’s messages to students without additional explanation, e.g., Ludwig [33], Ask-Elle [23], and ELP [24]. “When the student offers the program for analysis, the system [Ludwig] will first attempt to compile and link the program; if this step fails, then the compilation failure messages will be displayed to the student” [33]. Ask-Elle presents “a syntax-error message generated by the Helium compiler” to the student in case there is a compilation error [41]. Using ELP, “if there are syntax errors, compiler error messages are returned to the student” [42].

Instead of solely returning compiler’s feedback, VC PROLOG [39] provides detailed syntactic error explanation that is associated with a concept in the defined ontology of Prolog. For example, VC PROLOG displays the following explanations for syntax errors instead of showing the compiler’s output: “Probably a closing bracket is missing in this list” indicates that the student has forgotten a bracket; “A variable ‘X’ was written lower case, therefore it is interpreted as a constant” explains how the system interprets the lower case “x” ([39], p. 72). Gross and colleagues [43] argued that students are often not able to write programs on their own (e.g., on paper) and do not understand the cause of errors produced by a compiler. Thus, the authors proposed a new tutoring approach, which initiates a dialogue-based discourse between a student and an intelligent tutor in case of a syntactic error.

3.3. Semantic Feedback

Feedback on this level points to an error in the student's program with respect to fulfilling the requirements of a programming task. An educational system that supports semantic feedback does not necessarily presuppose a structured problem solving activity. Problem solving can take place in an exploratory learning manner. Feedback messages of this type can be divided into two levels: intention analysis and code analysis. Analysis on the intention level means identifying or hypothesizing the solution strategy intended by the student and giving feedback based on the identified and hypothesized solution strategy. Code analysis is meant identifying erroneous code, which does not fulfill the requirements of a given programming exercise. Some systems are able to provide both intention-based and code-based feedback that is referred to as two-level feedback. Other systems are only able to support only code-based feedback. In the following, these two types of semantic feedback are elaborated. Since semantic feedback results from the analysis of an individual student's program, semantic feedback can fall into one of the classes: explanatory feedback, diagnostic feedback, elaborative feedback in the classification of Fleming and Levie [16]; response contingent, bug-related, or topic contingent feedback in the classification of Economides [17]; or elaborated feedback in the classification of Narciss [7].

3.3.1. Two-Level Feedback: Intention-Based and Code-Based

On the intention-based level, Hong's PROLOG "tries to recognize programming techniques in the hierarchy of relevant programming techniques, which have been used in the student program" ([26], p. 519). Hong used grammar rules to model Prolog programming techniques, which serve to identify the intention (in terms of programming techniques) of the student. The system iteratively uses the sets of grammar rules to parse the student's program. If the parsing procedure does not finish successfully, that means, the selected set of grammar rules has not been completely exploited and the strategy of the student's program cannot be identified. In this case, the system uses one of the possible solution strategies (in terms of programming techniques) specified for a given programming problem to guide the student. Otherwise, the solution strategy has been identified, and the system diagnoses errors in the student's program. The system uses the same set of grammar rules, with which the solution strategy has been identified, to parse a corresponding reference program. For each possible solution strategy specified for a problem, there is a corresponding reference program. The parse tree of the student's program is compared against the parse tree of the reference program. Hong's PROLOG is able to analyze a student's program not only on the intention level, but also on the code level. The system's domain model contains not only domain's knowledge on the intention level (*i.e.*, sets of programming technique represented by grammar rules for each class of programs), but also coding-related knowledge, which is represented by reference programs for each anticipated programming technique. For example, the system returns the following code-based feedback to the student: "You have used an appropriate programming technique for this exercise. The base case in your program is, however, not quite right. It is supposed to say that reversing an empty list gets the empty list itself. The second argument of your base case, List, is a variable but it is supposed to be an empty list, []. Now can you correct your base case?" ([26], p. 523).

For JavaBugs [29], unfortunately, sample feedback messages were not available in literature. However, feedback may be reasoned from the following excerpt: "The task of automatic bug library construction entails detecting the most similar correct program (intention expressed as reference programs), extracting the superficial differences (discrepancies) between the student's and the correct program and forming misconception definitions (error hierarchies) described by discrepancies based on similarity and causality heuristics." ([29], p. 185). The authors applied a multi-strategy machine learning approach to automatically construct a library of Java errors, which novice programmers often make. Using this bug library, the developed educational system is able to examine "small" Java programs. The intention of the student is identified by comparing the student's program to a set of reference programs. After the student's intention has been identified, differences between the reference program and the student's solution are extracted. Misconceptions are learned based on the discrepancies identified in the first phase. These misconceptions are used to update the error library and to return appropriate feedback to the student.

Applying the similar multi-strategy machine learning approach for constructing a library of errors, while JavaBugs was developed to support Java learning, MEDD [34] is able to identify the student's intention in a Prolog program. Intentions are defined by specific reference programs. The similarity between the student's program and the reference programs is determined by matching surface as well as abstract features. Both JavaBugs and MEDD use a bug library, which is built automatically by learning from incorrect solutions in order to provide feedback on code analysis. Based on discrepancies between the incorrect student's program and its associated reference program, the systems provide semantic feedback on the code analysis level to students.

Ask-Elle [23] is able to derive the intention of the student using a set of pre-specified reference programs. Then, using the identified programming strategy of the reference program, many program variants are generated. All generated solutions are normalized before they are used to compare with the student's program. Through the comparison between the student's program and the generated program variants, the system not only has the ability of analyzing the student's intention underlying a program, but also is able to analyze the student's program on the code level. The system is able to provide different types of feedback based on the identified intention of the student. It can provide a description of a particular reference program, propose an implementation for a function, or emphasize one particular implementation method. For instance, Ask-Elle shows an elaborative feedback such as "Unexpected right hand side of f on line 3" ([41], p. 252), this feedback indicates the location of the error in the student's program (on the right hand side of the function f).

VC PROLOG is also able to provide semantic feedback by comparing between the student's program and one or more reference programs. Unfortunately, an example for intention-based feedback provided by VC PROLOG could not be found in literature. Only code-based feedback was presented: "A clause was terminated with a colon instead of a period, which causes the following fact to be interpreted as an addition goal." ([39], p. 72). The process of analyzing a student's program consists of three phases. First, a parser analyzes the student's program. In case there are syntax errors, they are marked and corrected. If there are several possible interpretations, the analyzer will produce all alternatives. Second, the syntactically correct student's program is semantically analyzed. That is, the meaning of the student's program is compared against the meaning of the reference program. Through this phase, the system is able to derive the intention of the student's strategy and to give feedback not only on the intention level, but also on the code analysis level. The third phase is analyzing the layout and the structure of the student's program based on the reference program.

INCOM [27] is also able to provide intention-based feedback: "INCOM favours to formulate feedback messages in terms of programming techniques to help students develop the skills of using high level programming concepts in logic programming." ([27], p. 106). The author of INCOM proposed a weighted constraint-based model (WCBM) to hypothesize the solution strategy intended in the student's program. First, on the strategy level, the system generates hypotheses about the student's intention by iteratively matching the student's solution against the solution strategies that are specified in the semantic table. Next, once a solution strategy has been matched, the process initiates hypotheses about the student's solution variant by matching components of the student's solution against corresponding components of the selected solution strategy. As the final step, hypotheses generated on the solution variant level are evaluated, and the most plausible variant of the student's solution (within a strategy) is chosen. Using the best hypothesis on the strategy level and the best hypothesis on the solution variant level, the errors in the student's solution are identified.

We note that many systems used programming techniques, correct solutions, specific exemplars, model solution, or example solutions to examine the semantic fulfillment of students' solutions. All these terms convey the notion of a reference program that captures semantic requirements of a programming exercise.

3.3.2. Code Level Feedback

There is another class of systems that only support code-based feedback without analyzing the intention of students. J-Latte provides feedback on the code-based level, for example, “All statements require a semi-colon at the end (to say the statement is finished)” ([31], p. 59). The system uses pre-specified constraints that represent the principles of the Java domain in order to check the correctness of students’ programs.

JITS [30], which is based on the ACT-R cognitive theory [44], is able to display erroneous code of “small” Java programs for a specific programming problem. Although JITS also analyzes the intention of the student using the Intent Recognition module first, then analyzes the programming code, this system only provides code-based feedback like “I think you meant the keyword ‘**for**’. Is this correct?” ([45], p. 617).

ELM-ART [14] provides exercises for problem solving. The domain knowledge of ELM-ART consists of a hierarchy of concepts and rules. Using this type of knowledge, the system is able to analyze the student’s program on the code level as the authors described: “The system gives feedback by providing a sequence of help messages with increasingly detailed explanation of the error or suboptimal solution.” ([46], pp. 368–369).

jTutors [32] allows instructors to author exercises in form of quizzes by specifying correct values and hints. Using these pre-specified correct values and hint messages, the system is able to test student’s knowledge by requesting him/her to fill the blanks in a quiz with correct values. If incorrect values are detected, instructor’s authored hint message will be provided. Or, “if the student struggles to answer a quiz, jTutors will provide an example or another quiz with lower difficulty level.” ([32], p. 12).

Radosevic and colleagues [38] developed a specific learning interface for C++. They introduced the notion of semaphore, which represents a number of new programming lines after the last compilation. Radosevic’s learning programming interface provides feedback not in textual form, rather using three colors: green, yellow, and red. Semaphore works in a way that each new programming line and each semicolon are enumerated. If a semaphore has 0–4 new lines, then the student’s program is inside green light. If the semaphore of the student’s program has 5–10 new lines, then it is yellow. If the light is red, the student’s program cannot be compiled and the student has to reduce number of lines. It is not clear that this type of feedback is adaptive because many students may get the same color (green, yellow or red) as feedback.

Goshi’s ITS for Hoare Logic [25] is able to provide appropriate advice or information on the code analysis level as the author stated: “This feedback may include appropriate advice or information in addition to indicating whether the answer supplied was correct or incorrect” [25]. This kind of advice of information is based on accessing the student’s behavior sequence.

JACK [28] follows a special way of providing feedback. The authors of JACK used automated trace generation for supporting students. A trace of a program execution is a list of single execution steps that are indicated by the system states in terms of variable values. The authors claimed that reading the trace of a program execution, a student could understand how a program works, *i.e.*, the system state evolves from the given input to the final state. However, students do not need to read a trace from the beginning to the end. A trace may indicate the erroneous location in the student’s program and the student needs to “search for program steps that directly affected the final erroneous output. This way, students can try to identify the reasons for the erroneous output and correct the related program statements” ([28], p. 304). This way, the trace of the student’s program execution is a kind of feedback. Since the trace is different for different students’ programs, feedback in form of traces is adaptive.

FIT Java Tutor [10] and ITAP [11] deployed the data-driven approach to diagnose errors in the student’s solution. That is, a mass of correct and incorrect students’ solutions for a programming problem is collected and used for analyzing a submitted student’s solution. FIT Java Tutor is able to provide code-based feedback by comparing the student’s solution with one of the pre-specified reference programs or students’ correct solutions. The students’ correct solutions have been collected in advance (maybe in previous courses) and should solve a given programming problem correctly. The students' solution attempts enhance the solution space of correct solutions for a programming problem. For example, the following feedback message indicates that the student's solution has been matched to one of the pre-specified reference programs or students' solution attempts: "The system has determined a certain similarity between your program and the program shown above. Compare the two programs and modify your program if necessary." ([10], p. 26). To provide automated feedback, the authors applied machine learning methods that compare the similarity between the student's solution and the pre-specified reference programs (or the collected students' correct solutions).

ITAP is able to provide code-based feedback, e.g., "in line 1 replace 'saturday' with 'Saturday' in the right side of the comparison operation" ([11], p. 17). In order to be able to provide feedback, the system requires two kinds knowledge: (1) at least one reference program for a given programming problem; and (2) a test method that can score, e.g., pairs of expected input and output of code. Based on the reference program, the system constructs a solution space for a problem. The authors defined a solution space as a graph of intermediate states that the student passes through as he/she works on a given problem. Each state is represented by the student's current code. The system starts the process of path construction by inserting new "correct" states into the solution space. The correct states are checked by the test methods. These "correct" states serve to define the optimal path of actions to solve a given problem and to generate feedback by recommending the action a student should take to move from their incorrect state to a correct one.

3.4. Layout Feedback

Layout feedback is intended to help students write a code according a specific coding convention so that the student himself/herself and peers can understand the student's code easier. The second purpose of layout feedback is to help students have a clear code so that they can find errors easier if their code is erroneous. Feedback on the layout level is only supported by few existing systems, e.g., Ludwig and VC PROLOG. Ludwig can perform stylistic analysis on a student's program. The stylistic analysis includes: proper indentation, cyclometric complexity, global variables, *etc.* The authors stated that the stylistic tests could be individually enabled by the instructor because not all instructors would need them ([33], p. 58). Whereas Ludwig focuses on stylistic analysis, VC PROLOG provides layout feedback by comparing the layout and the structure of the student's program with a reference program.

In principle, if there were a coding convention for every programming language, a component for checking style for that specific programming language can be developed and integrated into a learning environment. It may help students structure and debug their programs easily. Unfortunately, most of the systems reviewed in this paper did not implement layout analysis. A possible explanation for the rare support of layout feedback in educational systems for programming is that most researchers focus on instructing syntax and semantic issues of a programming language, while the layout of a program is a secondary issue.

3.5. Quality Feedback

ELP and JACK are the systems among the reviewed ones that support quality analysis based on the metrics for software engineering. Quality analysis focuses on how a program is executed efficiently by a corresponding machine model of a programming language. Quality analysis not only checks whether an algorithm works, but also whether that algorithm is efficient in terms of time and memory resource. For example, ELP is able to assess that the expression "if (a==true){}" is a poor programming practice and the expression "if (a){ }" is a good programming practice ([24], p. 114). ELP's quality analysis includes two phases: structural similarity analysis and software engineering analysis. Structural similarity analysis indicates the location of codes that have high complexity, lengthy, or poor programming practice by comparing the student's program with a set of reference programs. Software engineering analysis also serves to identify the location of complex or poor codes.

The author intended to apply both analysis approaches in order to enhance the analysis accuracy, *i.e.*, results of the structural similarity analysis is used to verify results of the software engineering analysis.

In addition to providing semantic feedback, JACK [28] is also able to provide quality analysis based on generated trace. The system compares the trace of a reference program with the trace of the student's program. If the trace of the student's program is longer than the reference program, "in this case, the solution seems to be unnecessarily complicated. The code executed during the trace steps without alignment are consequently most likely superfluous" ([28], p. 306). Thus, the length of the trace is an indicator for the quality of the student's program.

It seems that the issue of quality analysis is not the primary goal of the majority of programming novices. As a result, not many educational systems for programming focused on this issue.

4. Impact of Adaptive Feedback

In this section, the impact of adaptive feedback on supporting students in developing their programming skills is investigated. For this purpose, the evaluation studies that have been conducted for the systems were taken into account.

The objectives of the evaluation studies conducted for the educational systems for programming were very various. In general, two classes of evaluation objectives for those systems are distinguished: (1) pedagogical evaluation studies that investigated the impact of systems on learning on cognitive, meta-cognitive and motivational dimensions; and (2) technical evaluation studies that investigated technical properties of a system (e.g., regarding the algorithms' accuracy, validity of test cases, and system's performance) and its usability. For example, Hong evaluated his PROLOG Tutor regarding the recognition ability of programming techniques applied in students' solutions and error analysis based on identified programming techniques. MEDD's performance was evaluated regarding: (1) program classification; and (2) error discovery. JavaBugs' library for detecting bugs in students' Java programs was evaluated regarding its accuracy in identifying correct and incorrect students' programs, respectively (*cf*. Table 2). The methods for pedagogical evaluation studies that have been conducted for educational systems can be divided into quantitative and qualitative classes. Qualitative evaluation studies rate the systems by asking users' attitudes using questionnaires or conducting interviews. Quantitative evaluation studies are usually conducted in labors or in real class settings where students are requested to use to the educational systems. Data collected from interactions between the students and the systems serve to analyze the systems' effectiveness.

Table 2 lists educational systems for programming that have been reviewed in this paper. In the case that a pedagogical evaluation study for an educational system has been conducted and demonstrated positive results, then it is marked as "+" in the column "Result" and a quote is cited from the corresponding literature to prove this. If there was only technical evaluation for a system, no results with respect to learning effectiveness were available and thus, the column "Result" is empty. In addition, Table 2 also summarizes the feedback types that are supported by an educational system. "Y" indicates that a specific feedback type is supported. "I" and "C" stand for intention-based feedback and code-based feedback, respectively.

5. Summary, Related Work, and Research Directions

Five types of feedback that are supported by educational systems for programming have been identified: Yes/No feedback, syntax feedback, semantic feedback, layout feedback, and quality feedback. Semantic feedback can be divided into two levels: intention-based and code-based feedback. While Yes/No feedback and semantic feedback are in accordance with the generic feedback classifications [7,16,17], syntax feedback, layout feedback, and quality feedback are specific in the domain of programming. Thus, the classification of feedback types that have been identified in reviewed educational systems for programming can be clearly distinguished from those generic feedback classifications.

The first lesson we can learn is that there are various forms of feedback. Most systems support textual feedback, while few other systems provide feedback in special forms: e.g., execution trace of a program (e.g., JACK) or semaphores (red, yellow, and green, e.g., Radosevic's system). To the best of my knowledge, no study has been conducted to compare the effectiveness of different feedback forms yet.

With respect to feedback types, we can learn that numerous systems support semantic feedback, while only three systems (QuizJET, QuizPack, and Radosevic's system) provide Yes/No feedback for improving programming skills of students. Interestingly, there are several systems that support both semantic feedback and syntax feedback/layout feedback. Due to the specified selection criteria of the systems (*cf.* Section 2), any educational system for programming that supports syntax feedback must provide more information than a compiler, all reviewed systems that support syntax feedback must also provide semantic feedback, or layout feedback (e.g., VC PROLOG, Ludwig), or quality feedback (e.g., ELP). To the best of my knowledge, in literature no study has investigated the effectiveness of syntax feedback in isolation yet. Among the systems that support semantic feedback, several program analysis approaches (Ask-Elle, Hong's PROLOG, INCOM, JavaBugs, MEDD, VC PROLOG) are able to analyze the student's intention. Another open question to be investigated is whether systems that support intention-based feedback in addition to code-based feedback are better than systems that provide only feedback on the code level.

In order to diagnose errors in student's solutions and to provide appropriate feedback, various program analysis techniques have been developed: approach using pre-specified correct values (QuizJET, QuizPack, M-PLAT), syntax analysis using programming language compilers (Ludwig, Ask-Elle, ELP), approach using concept ontology (VC PROLOG), program analysis using grammar rules (Hong's PROLOG), approach using multi-strategy machine learning (JavaBugs, MEDD), approach using program transformation (Ask-Elle), approaches based on cognitive learning theories (model-tracing (JITS), constraint-based modeling (J-Latte), weighted constraint-based model (INCOM)), and data-driven approach (ITAP, FIT Java Tutor). An interesting finding is that although many various modeling techniques have been devised, each modeling technique has been deployed in a single system. That is, the developed modeling techniques do not have widespread use in practice or across several tools for programming education. Several reasons for this phenomenon can be derived. First, each developed modeling technique might be applicable for a specific programming concept or for a specific programming language, whereas the number of programming concepts of a programming paradigm is high and the number of programming languages is not predictable. Second, time required for devising a modeling technique for a programming language or a programming concept might be very high, so that researchers, after having tested a devised modeling technique, may not have human and time resources for testing the applicability of devised modeling techniques in a new programming language or with a new programming concept. One thing in common between the program analysis approaches is that they use a reference program to compare the difference between a reference program and a student's program (e.g., Ask-Elle, ELP, and Ludwig), or they deploy an abstract representation capturing required components to analyze students' programs (e.g., Hong's PROLOG, INCOM, and VC PROLOG). The reference program and the abstract representation of required components are required to understand a student's program and to detect errors in a student's program.

From the evaluation studies of the reviewed educational systems for programming (Table 2), we can learn that most systems have been evaluated and demonstrated their pedagogical effectiveness. To my best knowledge, no study has been conducted to compare the effectiveness of each feedback type nor has a between-system study been carried out to compare the effectiveness of adaptive feedback yet. This can be explained by the fact that the positive impact of an educational system is contributed by many features of the whole system: e.g., adaptive feedback, usability, and algorithms' accuracy.

The review being presented in this paper addresses adaptive feedback types that are supported by existing educational system for programming and this purpose has not been focused by other existing reviews.

In 1988, Ducassé and Emde [47] reviewed 18 environments and classified them into: (1) systems related to tutoring task; (2) general purposes debugging systems; and (3) enhanced Prolog tracers. Although the purpose of this review was to classify debugging knowledge and techniques deployed in these systems, there were only seven systems for educational purposes.

One decade later, Deek and McHugh [18] published a survey in 1998 that reviewed 29 computer-supported systems for learning programming and identified four categories of tools: (1) programming environments; (2) debugging aids; (3) intelligent tutoring systems; and (4) intelligent programming environments. Since the detailed survey of Deek and McHugh [18], there were several other surveys on computer-supported systems for learning programming, however, most of them focused on specific aspects. For instance, Deek and colleagues [48] reviewed web-based environments for program development. Guzdial [49] investigated programming environments for novices. Douce and colleagues [50] reviewed computer-supported systems that automatically assess students' programming assignments. While these surveys focused on intelligent features (e.g., program analysis and assessment of programming assignments) of systems for supporting students, it is worth noting, on the contrary, Gomez-Albarran [51] focused their survey on less sophisticated and less intelligent approaches. That is, those approaches to learning/teaching programming do not support automatic program analysis nor automatic diagnosis of errors in student's programs. Gomez-Albarran reviewed nearly 20 "most outstanding" tools of the following types: (1) tools including a reduced development environment (e.g., BlueJ [52]); (2) example-based environments (e.g., ELM-ART [14]); (3) tools based on visualization and animation (e.g., ANIMAL [53]); and (4) simulation environments (e.g., Karel++ [54]). In another survey, Le and colleagues [55] reviewed AI-supported instructional approaches for learning programming and classified them into: example-based, simulation-based, collaboration-based, dialogue-based, program analysis-based, and feedback-based approaches.

The review being presented in this paper focuses on the aspect of adaptive feedback in educational systems for programming. It is now the right time to investigate feedback types and analysis techniques that have been deployed in existing educational systems for programming due to two reasons. First, feedback is one of the most important features of educational systems (especially for the domain of programming). It is used by programming learners to revise a solution for a programming problem. Second, over the last three decades, not only have new systems been developed, new program analysis approaches have also been proposed to enhance the diagnosis accuracy, while most conducted surveys of educational systems for programming rather focused on classifying the systems ([18,47,51]).

The classification of adaptive feedback that has been developed in this review serves researchers and developers of educational systems for programming two purposes. First, it helps researchers and developers apply and advance existing programming analysis techniques that have been evaluated and validated. They may enhance these analysis techniques by optimizing the analysis accuracy. Second, the classification of adaptive feedback provides researchers and developers a means to focus on specific feedback types when developing an educational system for programming that is intended to support adaptive feedback.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Orsini, L. Why Programming Is the Core Skill of the 21st Century? 2013. Available online: http://readwrite.com/2013/05/31/programming-core-skill-21st-century (accessed on 13 January 2015).
2. Bellaby, G.; McDonald, C.; Patterson, A. Why lecture? In Proceedings of the 3rd Conference of the LTSN Centre for Computer and Information Science, Loughborough, UK, 27–29 August 2002; pp. 228–231.
3. Lahtinen, E.; Ala-Mutka, K.; Järvinen, H.-M. A study of the difficulties of novice programmers. *SIGCSE Bull.* **2005**, *37*, 14–18. [CrossRef]
4. Baum, W.M. *Understanding Behaviorism: Behavior, Culture and Evolution*; Blackwell: Oxford, UK, 2005.
5. Wiener, N. *Cybernetics, or Control and Communication in the Animal and the Machine*; MIT Press: Cambridge, MA, USA, 1948.

6. Karkalas, S.; Gutierrez-Santos, S. Enhanced javascript learning using code quality tools and a rule-based system in the FLIP exploratory learning environment. In Proceedings of the 14th International Conference on Advanced Learning Technologies (ICALT), Athens, Greece, 7–10 July 2014.
7. Narciss, S. Designing and Evaluating Tutoring Feedback Strategies for Digital Learning Environments on the Basis of the Interactive Tutoring Feedback Model. Digital Education Review—Number 23, 2013.
8. Dempsey, J.V.; Sales, G.C. *Interactive Instruction and Feedback*; Educational Technology Publications: Englewood Cliffs, NJ, USA, 1993.
9. Gouli, E.; Gogoulou, A.; Papanikolaou, K.; Grigoriadou, M. An Adaptive Feedback Framework to Support Reflection, Guiding and Tutoring. In *Advances in Web-based Education: Personalized Learning Environmentss*; Magoulas, G., Chen, S., Eds.; Idea Group Inc.: Calgary, AB, Canada, 2006; pp. 178–202.
10. Gross, S.; Pinkwart, N. Towards an integrative learning environment for Java programming. In Proceedings of the IEEE 15th International Conference on Advanced Learning Technologies (ICALT), Hualien, Taiwan, 6–9 July 2015; pp. 24–28.
11. Rivers, K.; Koedinger, K.R. Data-driven hint generation in vast solution spaces: A self- improving python programming tutor. *Int. J. Artif. Intell. Educ.* **2016**. [CrossRef]
12. Narciss, S.; Sosnovsky, S.; Schnaubert, L.; Andrès, E.; Eichelmann, A.; Goguadze, G.; Melis, E. Exploring feedback and student characteristics relevant for personalizing feedback strategies. *Comput. Educ.* **2014**, *71C*, 56–76. [CrossRef]
13. Brusilovsky, P.; Millán, E. User models for adaptive hypermedia and adaptive educational systems. In *The Adaptive Web*; Brusilovsky, P., Kobsa, A., Nejdl, W., Eds.; Springer-Verlag: Berlin, Germany, 2007; pp. 3–53.
14. Weber, G.; Brusilovsky, P. ELM-ART—An interactive and intelligent web-based electronic textbook. *Int. J. Artif. Intell. Educ.* **2015**, *26*, 72–81. [CrossRef]
15. Arroyo, I.; Beck, J.E.; Beal, C.R.; Wing, R.; Woolf, B. Analyzing students' response to help provision in an elementary mathematics Intelligent Tutoring System. In Proceedings of the Workshop on Help Provision and Help Seeking in Interactive Learning Environments held at the 10th International Conference on Artificial Intelligence in Education, San Antonio, TX, USA, 19–23 May 2001; pp. 34–46.
16. Fleming, M.; Levie, W. *Instructional Message Design: Principles from the Behavioral and Cognitive Sciences*; Educational Technology Publications: Englewood Cliffs, NJ, USA, 1993.
17. Economides, A.A. Adaptive feedback evaluation. In Proceedings of the 5th WSEAS International Conference on Distance Learning and Web Engineering, Corfu, Greece, 23–25 August 2005; pp. 134–139.
18. Deek, F.P.; McHugh, J.A. A survey and critical analysis of tools for learning programming. *Comput. Sci. Educ.* **1998**, *8*, 130–178. [CrossRef]
19. Lane, H.C.; VanLehn, K. Teaching the tacit knowledge of programming to novices with natural language tutoring. *Comput. Sci. Educ.* **2005**, *15*, 183–201. [CrossRef]
20. Kynigos, C. Half-baked Logo microworlds as boundary objects in integrated design. *Inform. Educ. Int. J.* **2007**, *6*, 335–358.
21. Morgado, L.; Kahn, K. Towards a specification of the ToonTalk language. *J. Vis. Lang. Comput.* **2008**, *19*, 574–597. [CrossRef]
22. Dasgupta, S.; Resnick, M. Engaging novices in programming, experimenting, and learning with data. *ACM Inroads* **2014**, *5*, 72–75. [CrossRef]
23. Gerdes, A.; Heeren, B.; Jeuring, J.; van Binsbergen, L.T. Ask-Elle: An Adaptable Programming Tutor for Haskell Giving Automated Feedback. *Int. J. Artif. Intell. Educ.* **2016**. [CrossRef]
24. Truong, N. A Web-Based Programming Environment for Novice Programmers. Ph.D. Thesis, Queensland University of Technology, Brisbane, Australia, 2007.
25. Goshi, K.; Wray, P.; Sun, Y. An intelligent tutoring system for teaching and learning Hoare logic. In Proceedings of the 4th IEEE International Conference on Computational Intelligence and Multimedia Applications, Yokusika, Japan, 30 October–1 November 2001.
26. Hong, J. Guided programming and automated error analysis in an intelligent Prolog tutor. *Int. J. Hum. Comput. Stud.* **2004**, *61*, 505–534. [CrossRef]
27. Le, N.-T. Using Weighted Constraints to Build a Tutoring System for Logic Programming. Ph.D. Thesis, University of Hamburg, Hamburg, Germany, 2011.

28. Striewe, M.; Goedicke, M. Using run time traces in automated programming tutoring. In Proceedings of the 16th ACM Joint Conference on Innovation and Technology in Computer Science Education, Darmstadt, Germany, 27–29 June 2011; pp. 303–307.
29. Suarez, M.; Sison, R. Automatic construction of a bug library for object-oriented novice java programmer errors. In Proceedings of the 9th Conference on Intelligent Tutoring Systems, Montreal, QC, Canada, 23–27 June 2008; Springer: Berlin, Germany, 2008; pp. 184–193.
30. Sykes, E.R. Qualitative evaluation of the java intelligent tutoring system. *J. Syst. Cybern. Inform.* **2006**, *3*, 49–60.
31. Holland, J. A Constraint-Based ITS for the Java Programming Language. Master's Thesis, University of Canterbury, Christchurch, New Zealand, 2009.
32. Dahotre, A. jTutors: A Web-Based Tutoring System for Java APIs. Master's Thesis, Oregon State University, Corvallis, OR, USA, 2011.
33. Shaffer, S.C. Ludwig: An online programming tutoring and assessment system. *SIGCSE Bull.* **2005**, *37*, 56–60. [CrossRef]
34. Sison, R.C.; Numao, M.; Shimura, M. Multistrategy discovery and detection of novice programmer errors. *J. Mach. Learn.* **2000**, *38*, 157–180. [CrossRef]
35. Nunez, A.; Fernandez, J.; Garcia, J.D.; Prada, L.; Carretero, J. M-PLAT: Multi-programming language adaptive tutor. In Proceedings of the 2008 Eighth IEEE International Conference on Advanced Learning Technologies, Santander, Spain, 1–5 July 2008; pp. 649–651.
36. Hsiao, I.-H.; Sosnovsky, S.; Brusilovsky, P. Adaptive navigation support for parameterized questions in object-oriented programming. In Proceedings of the 4th European Conference on Technology Enhanced Learning (ECTEL), Cannes, France, 29 September–2 October 2009; Springer: Berlin, Germany, 2009; pp. 88–98.
37. Brusilovsky, P.; Sosnovsky, S. Individualized exercises for self-assessment of programming knowledge: An evaluation of QuizPACK. *ACM J. Educ. Resour. Comput.* **2005**, *5*, 6. [CrossRef]
38. Radosevic, D.; Lovrencic, A.; Orehovacki, T. New approaches and tools in teaching programming. In Proceedings of Central European Conference on Information and Intelligent Systems, Varaždin, Croatia, 23–25 September 2009.
39. Peylo, C.; Thelen, T.; Rollinger, C.; Gust, H. A web-based intelligent educational system for prolog. In Proceedings of the International Workshop on Adaptive and Intelligent Web-Based Education Systems, Montreal, QC, Canada, 19–23 June 2000.
40. Nunez, A.; Fernandez, J.; Carretero, J. M-PLAT: Multi-programming language adaptive tutor. In *Advances in Learning Processes*; Rosso, M.B., Ed.; InTech: West Palm Beach, FL, USA, 2010.
41. Gerdes, A.; Jeuring, J.; Heeren, B. An interactive functional programming tutor. In Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education, Haifa, Israel, 3–5 July 2012; pp. 250–255.
42. Truong, N.; Bancroft, P.; Roe, P. A web based environment for learning to program. In Proceedings of the 26th Australasian Computer Science Conference, Adelaide, Australia, 4–7 February 2003; Volume 16, pp. 255–264.
43. Gross, S.; Strickroth, S.; Pinkwart, N.; Le, N.-T. Towards deeper understanding of syntactic concepts in programming. In Proceedings of the 16th International Conference on Artificial Intelligence in Education Part 9: The First Workshop on AI-Supported Education for Computer Science (AIEDCS), Memphis, TN, USA, 9–13 July 2013.
44. Anderson, J.R.; Boyle, C.F.; Corbett, A.T.; Lewis, M.W. Cognitive modelling and intelligent tutoring. *Artif. Intell.* **1990**, *42*, 7–49. [CrossRef]
45. Sykes, E.R.; Franek, F. Inside the Java intelligent tutoring system prototype: Parsing student code submissions with intent recognition. In Proceedings of the IASTED International Conference Web-Based Education, Innsbruck, Austria, 16–18 February 2004.
46. Weber, G.; Brusilovsky, P. ELM-ART: An adaptive versatile system for web-based instruction. *Int. J. Artif. Intell. Educ.* **2001**, *12*, 351–384.
47. Ducassé, M.; Emde, A.-M. A review of automated debugging systems: Knowledge, strategies and techniques. In Proceedings of the 10th International Conference on Software Engineering, 11–15 April 1988; pp. 162–171.
48. Deek, F.; Ho, K.-W.; Ramadhan, H. A review of web-based learning systems for programming. In Proceedings of the World Conference on Educational Multimedia, Hypermedia and Telecommunications, Tampere, Finland, 25–30 June 2001; Montgomerie, C., Viteli, J., Eds.; pp. 382–387.

49. Guzdial, M. Programming environments for novices. In *Computer Science Education Research*; Fincher, S., Petre, M., Eds.; Taylor & Francis: Abingdon, UK, 2004; pp. 127–154.
50. Douce, C.; Livingstone, D.; Orwell, J. Automatic test-based assessment of programming: A review. *J. Educ. Resour. Comput.* **2005**, *5*, 4. [CrossRef]
51. Gomez-Albarran, M. The teaching and learning of programming: A survey of supporting software tools. *Comput. J.* **2005**, *48*, 131–144. [CrossRef]
52. Barnes, D.J.; Kölling, M. *Objects First with Java*; Pearson Education: New York, NY, USA, 2012.
53. Rösling, G.; Freisleben, B. Animal: A system for supporting multiple roles in algorithm animation. *J. Vis. Lang. Comput.* **2002**, *13*, 341–354. [CrossRef]
54. Bergin, J.; Roberts, J.; Pattis, R.; Stehlik, M. *Karel++: A Gentle Introduction to the Art of Object-Oriented Programming*; John Wiley & Sons, Inc.: New York, NY, USA, 1996.
55. Le, N.-T.; Strickroth, S.; Gross, S.; Pinkwart, N. A review of AI-supported tutoring approaches for learning programming. In Proceedings of the 1st International Conference on Computer Science, Applied Mathematics and Applications (ICCSAMA), Warsaw, Poland, 9–10 May 2013; Springer Verlag: Berlin, Germany, 2013; pp. 267–279.